UNDERSTANDING AND IMPROVING NEURAL ARCHITECTURE SEARCH

SHU YAO

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2022

Supervisors:

Associate Professor Kian Hsiang Low, Main Supervisor

Former Assistant Professor Wang Wei, Former Supervisor

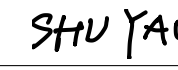
Examiners:

Associate Professor Lee Gim Hee

Assistant Professor Yao Yingjie Angela

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.



Shu Yao

August 28, 2022

Acknowledgements

Since I started my Ph.D. journey in 2017, four years and a half have passed silently. As expected, this journey is not easy to complete. There were uncountable days and nights when I was asking myself about how to make my Ph.D. journey more unique and meaningful. Surely, it was never an easy thing to reach the final answer to this question. Having gone through all the challenges along the way, I finally find that the answer lies in following the heart. Looking back, everything I did following from my inner desire always makes my Ph.D. journey more precious. I hope this memorable journey will always remind me to remain true to my original aspiration in the future.

I always feel that I am pretty fortunate to receive help and encouragement from many kind people I met along my Ph.D. journey. These help and encouragement make my life easier as always, which I appreciate. Especially, I would like to thank Prof. He Kun for taking me into the wonderful world of artificial intelligence. I am also grateful for the suggestions and supports received from my friends, e.g., Shaofeng, Zhongxiang. Particularly, I would like to express my sincere gratitude to my supervisor Prof. Bryan Kian Hsiang Low for his valuable suggestions, support and trust. With these, I can devote myself to my targets and dreams. I also would like to thank the entire MapleCG group for inspiring me constantly. I want to thank my elder sister for taking care

of our parents and giving me moral support when I was far abroad. Finally, I feel grateful for my education received from my parents, making everything that happened to me possible.

Contents

Abstract

Over the past decade, various famous *deep neural network* (DNN) architectures (He et al., 2016; Huang et al., 2017b; Simonyan and Zisserman, 2015; Krizhevsky et al., 2012) have been devised and have achieved superhuman performance for a wide range of tasks. Designing these neural networks, however, typically incurs substantial efforts from domain experts by trials and errors. Such human efforts gradually become unaffordable with an increasing demand for customizing DNNs for different tasks. To this end, *neural architecture search* (NAS) has been widely applied to automate the design of neural networks in recent years. In the literature, a number of NAS algorithms have been proposed, aiming to further improve the search efficiency and effectiveness of NAS, i.e., to reduce the search cost and improve the generalization performance of the selected architectures, respectively. Despite these advances, there are still certain essential aspects in NAS that have not been well investigated in the literature, which however may help us to understand and even further improve popular NAS algorithms.

Firstly, only a few efforts have been devoted to understanding the neural architectures selected by popular NAS algorithms in the literature. In the first work of this thesis (see Chapter 4), we take the first step of understanding popular NAS algorithms by answering the following questions: *What types of architectures are selected by popular NAS algorithms*

and why they are selected? In particular, we reveal that existing NAS algorithms (e.g., DARTS, ENAS) tend to favor architectures with wide and shallow cell structures. These favorable architectures consistently achieve fast convergence and are consequently selected by NAS algorithms. Our empirical and theoretical studies further confirm that their fast convergence derives from their smooth loss landscape and accurate gradient information. Nonetheless, these architectures may not necessarily lead to better generalization performance than other candidate architectures in the same search space, and therefore further improvement is possible by revising existing NAS algorithms.

Secondly, standard NAS algorithms typically aim to select only a single neural architecture from the search spaces and thus have overlooked the capability of other candidate architectures in helping improve the performance of their final selected architecture. To this end, we present two novel sampling algorithms under our *Neural Ensemble Search via Bayesian Sampling* (NESBS) framework that can effectively and efficiently select a well-performing ensemble of neural architectures from NAS search space (see Chapter 5). Compared with state-of-the-art NAS algorithms and other well-known ensemble search baselines, our NESBS algorithms are shown to be able to achieve improved performance in both classification and adversarial defense tasks on various benchmark datasets while incurring a comparable search cost to these NAS algorithms.

Thirdly, the search efficiency of popular NAS algorithms in the literature is severely limited by the need for model training during the search process. To overcome this limitation, in Chapter 6, we propose a novel NAS algorithm called *NAS at Initialization* (NASI) that exploits the capability of *Neural Tangent Kernel* (NTK) in being able to characterize the converged performance of candidate architectures at initialization, hence

allowing model training to be completely avoided to boost the search efficiency. Besides the improved search efficiency, NASI also achieves competitive search effectiveness on various datasets like CIFAR-10/100 and ImageNet. Further, NASI can guarantee the benefits of being *label-* and *data-agnostic* under mild conditions, i.e., the provable transferability of architectures selected by our NASI over different datasets.

Finally, though recent NAS algorithms using training-free metrics are able to select well-performing architectures in practice, the reason *why training-free NAS using these metrics performs well* and the answer to the question of *how training-free NAS can be further boosted* still have not been fully understood. To this end, we provide a unified theoretical analysis for gradient-based training-free NAS in this paper to understand why training-free metrics work well in practice (see Chapter 7). By exploiting these theoretical understandings, we then develop a novel NAS framework called *Hybrid Neural Architecture Search* (HNAS) that consistently improves training-free NAS in a principled way. Remarkably, HNAS can enjoy the advantages of both training-free (i.e., the superior search efficiency) and training-based NAS (i.e., the remarkable search effectiveness), which we have demonstrated through extensive experiments.

List of Works

Below is a list of works whose contributions are included in this thesis:

- **Shu, Y.**, Wang, W., & Cai, S. (2020). Understanding architectures learnt by cell-based neural architecture search. In *Proc. ICLR-20*.
- **Shu, Y.**, Chen, Y., Dai, Z., & Low, B.K.H. (2022). Neural Ensemble Search via Bayesian Sampling. In *Proc. UAI-22*.
- **Shu, Y.**, Cai, S., Dai, Z., Ooi, B. C., & Low, B. K. H. (2022). NASI: Label- and Data-agnostic Neural Architecture Search at Initialization. In *Proc. ICLR-22*.
- **Shu, Y.**, Dai, Z. Wu, Z., & Low, B.K.H. (2022). Unifying and Boosting Gradient-Based Training-Free Neural Architecture Search. Under review of *NeurIPS-22*.

Listed below are some other published works:

- Cai, S., **Shu, Y.**, & Wang, W. (2021). Dynamic routing networks. In *Proc. WACV-20*.
- Wu, Z., **Shu, Y.**, & Low, B.K.H. (2022). DAVINZ: Data Valuation using Deep Neural Networks at Initialization. In *Proc. ICML-22*.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

In recent years, we have witnessed the remarkable achievement that *deep learning* (DL) has made in a number of applications, such as in *computer vision* (CV) and *natural language processing* (NLP). Despite such compelling performance achieved by DL techniques, the design of neural networks is known to be a significant challenge repeatedly faced by DL practitioners while applying DL to various practical tasks due to the huge performance gap among different neural architectures like ResNet (He et al., 2016) vs. VGG (Simonyan and Zisserman, 2015). In the literature, various neural architectures (He et al., 2016; Huang et al., 2017b; Krizhevsky et al., 2012; Simonyan and Zisserman, 2015) have been devised by human experts over the past decades and achieved superhuman performance for a wide range of tasks. However, numerous human trials and errors need to be devoted to the design of neural architectures, which makes it prohibitively costly. As a consequence, the increasing demand for developing neural architectures for different tasks becomes unaffordable nowadays for those human experts. This therefore calls for

automated design of neural architectures, which has recently become an important topic in *automated machine learning* (AutoML) (Hutter et al., 2019).

To this end, *neural architecture search* (NAS) has been introduced to automate the design of neural architectures for different tasks by (Zoph and Le, 2017). As summarized in (Elsken et al., 2019a), NAS conventionally consists of a search space, a search algorithm, and a performance evaluation. Specifically, the search algorithm aims to select the best-performing neural architecture from the search space based on its evaluated performance via performance evaluation. Since (Zoph and Le, 2017), various search algorithms (Liu et al., 2018; Luo et al., 2018a; Real et al., 2019b; Zoph et al., 2018) have been proposed to select neural architectures with an improved generalization performance or search efficiency. Despite the advantages of achieving state-of-the-art (SOTA) performance with slightly improved search efficiency by these algorithms, they remain computationally costly due to the expensive model training of various candidate architectures during the search process, which hence becomes impractical for DL practitioners who are in shortage of computing resources. To overcome such a limitation, one-shot NAS has been widely adopted in recently proposed NAS algorithms (Dong and Yang, 2019b; Liu et al., 2019; Pham et al., 2018; Xie et al., 2019b). In particular, only the model training of one single supernet (namely, the one-shot architecture) is required and the performance of candidate architectures in the search space is estimated with the model parameters shared by this one-shot architecture. Consequently, the search efficiency of NAS has been improved significantly, thus leading to practical NAS for most DL practitioners.

Despite recent progress, there are still some important aspects of NAS

that need to be investigated, which can potentially help to understand or further improve current SOTA NAS algorithms. Specifically, we identify the following important aspects of NAS:

(a) There are only limited efforts being devoted to understanding the neural architectures selected by popular NAS algorithms in the literature. However, the investigation of the selected architectures by popular NAS algorithms could be significant for the whole NAS area because it may help us to understand *what types of architectures are selected by popular NAS algorithms and why they are selected*. By understanding these two questions, the limitations of existing NAS algorithms may be unveiled, which even may motivate or suggest possible remedies in the future. Moreover, the answers to these questions may also help the domain experts to better design the search space for NAS, i.e., to design the search space that covers high-potential architectures as well as its variants.

(b) Existing NAS algorithms usually select only *a single architecture* achieving the best performance in the search spaces and hence have ignored other candidate architectures in the search space that can help further improve the performance of their final selected architecture via neural network ensemble. As known, neural network ensembles are widely shown to be capable of achieving an improved performance compared with a single neural network on various tasks (Gal and Ghahramani, 2016; Cortes et al., 2017; Lakshminarayanan et al., 2017) in the literature. This naturally leads to an interesting question for the NAS area: *How can we select the best-performing ensemble of architectures from the NAS search space?* In the literature, only limited efforts (Zaidi et al., 2020) have been devoted

to the research topic of neural ensemble search. However, Zaidi et al. (2020) requires independent model training for all of their sampled architectures from the search space to conduct their neural ensemble search among these architectures. As a result, they have failed to explore the whole search space and thus have achieved poor ensemble performances in practice while still incurring unaffordable search costs for their independent model training during the search process.

(c) The search efficiency of popular NAS algorithms in the literature is severely limited by the need for model training during the search process, i.e., the model training of candidate architectures or the one-shot architecture. This naturally leads to the question *whether NAS is realizable at initialization such that model training can be completely avoided during the search process.* To the best of our knowledge, only a few efforts to date have been devoted to developing NAS algorithms without model training empirically (Mellor et al., 2020a; Park et al., 2020; Abdelfattah et al., 2021; Chen et al., 2021). Unfortunately, these works typically fail to provide theoretical support for their training-free methods or the transferability of their final selected architectures. Moreover, certain works even fail to achieve state-of-the-art results in large-scale experiments, which is typically required by the NAS area in order to examine the validity of the proposed NAS algorithms in practice.

(d) Though empirical results show that recent NAS using various training-free metrics is capable of finding well-performing architectures, the reason *why NAS using these training-free metrics performs well* in practice still has not been fully understood from a unified theoreti-

cal perspective. To the best of our knowledge, only limited efforts to date have been devoted to studying certain special training-free NAS forms theoretically, e.g., (Shu et al., 2021). Though the training-free NAS form in (Shu et al., 2021) can be derived from the training performance of candidate architectures, there remains a theoretical gap between training and generalization (i.e., validation/test) performance in their derivation. Moreover, a unified theoretical study for NAS based on training-free metrics may even inspire further unified improvement on training-free NAS and allow training-free NAS to be a practical alternative to training-based NAS.

1.2 Contributions

In this section, we give a brief summary of our contributions to understanding and improving current NAS algorithms following the aforementioned aspects in this thesis.

Understanding architectures learnt by cell-based neural architecture search. As nearly no effort has been devoted to understanding *what types of architectures are selected by popular NAS algorithms and why they are selected*, our work in Chapter 4 aims to empirically and theoretically answer these two questions. Specifically, by investigating the connection topologies of architectures selected by popular NAS algorithms (e.g., DARTS, ENAS), we figure out that these training-based NAS algorithms tend to select architectures with wide and shallow connection topologies. Interestingly, we also find that these wide and shallow connection topologies are usually able to achieve fast convergence (i.e., better generalization performance under the same training budget) and consequently will be selected by standard training-based NAS algorithms. After that, we then

provide both empirical and theoretical studies to further show that such a fast convergence actually derives from the smooth loss landscape and accurate gradient information of wide and shallow connection topologies. Finally, we show that these wide and shallow connection topologies may not achieve better generalization performance than other architectures in the search space, which therefore implies that further improvement on existing NAS algorithms is still possible and remains to be solved in the near future.

Neural Ensemble Search via Bayesian Sampling. According to (Zhou, 2012), both competitive single-model performances and diverse model predictions are essential for ensemble models to achieve compelling performance in practice. Inspired by this, our work in Chapter 5 introduces two novel sampling algorithms, i.e., *Monte Carlo sampling* (MC Sampling) and *Stein Variational Gradient Descent with regularized diversity* (SVGD-RD), under our *Neural Ensemble Search via Bayesian Sampling* (NESBS) framework, which are capable of sampling neural network ensembles with both competitive single-model performances and diverse model predictions and consequently can select well-performing ensembles of architectures effectively and efficiently from the NAS search space. Extensive empirical results show that our algorithms can achieve improved search effectiveness and efficiency than other state-of-the-art NAS, ensemble, and ensemble search baselines in both classification and adversarial defense tasks on various benchmark datasets while only incurring a comparable search cost to the standard NAS baselines.

NASI: Label- and Data-agnostic Neural Architecture Search at Initialization. Standard training-based NAS algorithms usually require the model training of candidate architectures or one-shot architecture in

their search process. Our work in Chapter 6 therefore presents a novel NAS algorithm called *NAS at Initialization* (NASI) to completely avoid the model training during the search process by employing the theory of *Neural Tangent Kernel* (NTK) to approximately characterize the performance of candidate architectures with only initialized parameters. Besides the improved search efficiency by conducting NAS at initialization, NASI is even proven to be *label-* and *data-agnostic* under mild conditions. This therefore guarantees the transferability of the architectures selected by our NASI over different datasets, i.e., the architectures selected by our NASI on one dataset may also perform well on other datasets with a high probability. Our extensive empirical results have shown that NASI is capable of achieving superior search efficiency (i.e., search cost), competitive search effectiveness (i.e., the performance of the final selected architectures) as well as compelling transferability on various benchmark datasets and search spaces. Finally, to summarize, our NASI significantly advances the line of training-free NAS in (a) providing theoretically grounded performance estimation by NTK (compared with (Abdelfattah et al., 2021; Chen et al., 2021; Mellor et al., 2020a)), (b) guaranteeing the transferability of its selected architectures with its provable *label-* and *data-agnostic* search under mild conditions (compared with (Abdelfattah et al., 2021; Chen et al., 2021; Mellor et al., 2020a; Park et al., 2020))) and (c) achieving SOTA performance in a large search space over various benchmark datasets (compared with (Abdelfattah et al., 2021) and (Mellor et al., 2020a; Park et al., 2020)).

Unifying and Boosting Gradient-Based Training-Free Neural Architecture Search. Even though empirical results show that NAS using various training-free metrics is capable of finding well-performing ar-

chitectures, the reason *why NAS using these training-free metrics performs well* in practice still has not been fully understood. To this end, our work in Chapter 7 provides unified theoretical analyses and improvement for *gradient-based* training-free NAS. Specifically, our unified analysis based on the provable connections among gradient-based training-free metrics firstly shows that these training-free metrics are theoretically related to the generalization performance of DNNs. Consequently, the performance of commonly applied training-free NAS forms based on these metrics can be theoretically guaranteed, which helps to understand the compelling empirical results achieved by them in practice. Besides the generalization guarantees, our analysis further indicates that the transferability in training-free NAS is also guaranteed. More interestingly, we even find that training-free NAS has the same preference of architecture topology as those training-based NAS algorithms in order to achieve their good final performance. Based on these theoretical analyses, we then develop a novel NAS framework named *Hybrid Neural Architecture Search* (HNAS) that enjoys the advantages of both training-based (i.e., the remarkable search effectiveness) and training-free NAS (i.e., the superior search efficiency), allowing training-free NAS to be further improved. Empirical results indicate that our theoretical analyses, as well as the effectiveness and efficiency of our HNAS can be well-supported in practice.

Overview of the contributions. Fig. 1.1 gives an overview of the contributions in this thesis. Specifically, our first work in Chapter 4 (i.e., Understanding NAS in Fig. 1.1) provides the missing interpretations for training-based NAS algorithms (aforementioned aspect (a) in NAS). Then, our second work in Chapter 5 (i.e., NESBS in Fig. 1.1) further improves the search effectiveness, e.g., the precision and performance, of training-

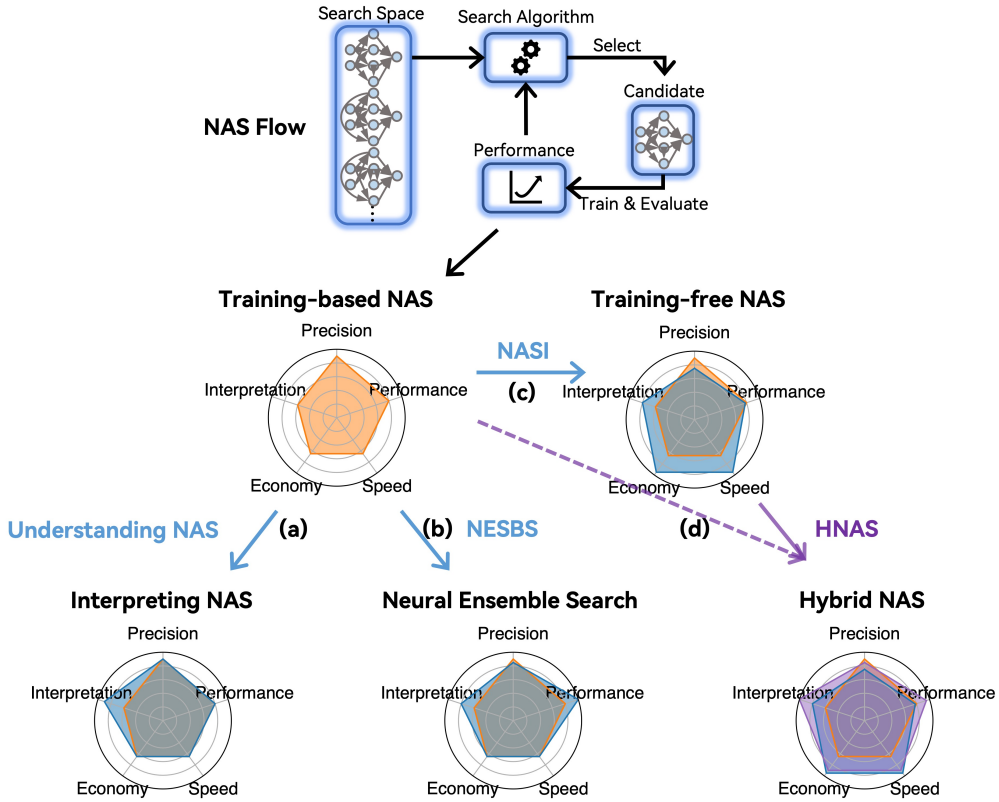


Figure 1.1: An overview of the contributions in this thesis.

based NAS (aforementioned aspect (b) in NAS). To significantly improve the search efficiency (e.g., the speed and economy,) of training-based NAS, our third work in Chapter 6 (i.e., NASI in Fig. 1.1) turns to NAS using training-free metrics for performance estimation (aforementioned aspect (c) in NAS). Finally, our last work in Chapter 7 (i.e., HNAS in Fig. 1.1) provides unified theoretical study for gradient-based training-free NAS and also unified improvement on its search effectiveness, e.g., the precision and performance, by integrating training-based NAS with training-free NAS (aforementioned aspect (d) in NAS).

1.3 Organization

In the remaining of this thesis, we firstly introduce the necessary background and notations detailed in Chapter 2. Chapter 3 then provides

an overview of the related works for the four works included in this thesis. Next, the following four chapters (i.e., Chapter 4,5,6,7) present each of our four works in detail. Finally, we complete this thesis with a conclusion and a future outlook in Chapter 8.

Chapter 2

Background and Notations

In this chapter, we introduce the necessary background and notations that will be useful throughout this thesis. Specifically, we will introduce *neural architecture search* (NAS) in Section 2.1 as a general background of this thesis. Then, the convergence of *stochastic gradient descent* (SGD) will be provided in Section 2.2, which serves as a theoretical foundation for our first work in Chapter 4. Next, we will introduce *Stein Variational Gradient Descent* (SVGD), which helps us to sample good ensemble candidates for our work in Chapter 5. Finally, we will detail *Neural Tangent Kernel* (NTK) in Section 2.4 that is exploited in our third and last work (see Chapter 6 and Chapter 7). As for the notations, we use lower-case bold-faced symbols to denote column vectors (e.g., $\boldsymbol{x}$), and upper-case bold-faced symbols to represent matrices (e.g., $\mathbf{X}$). Scalars are denoted by normal lower-case symbols, i.e., without bold highlight (e.g., x).

2.1 Neural Architecture Search

Similar to automated *hyper-parameter optimization* (HPO) that selects hyper-parameters with the best generalization performance, NAS intends to automatically find neural architectures achieving the best general-

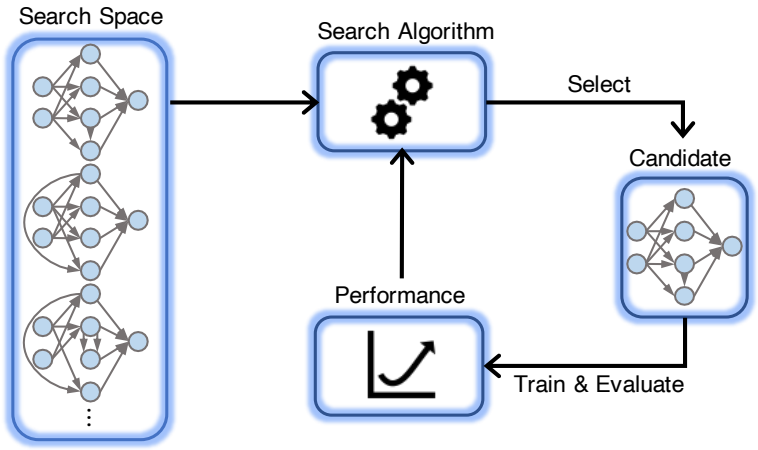


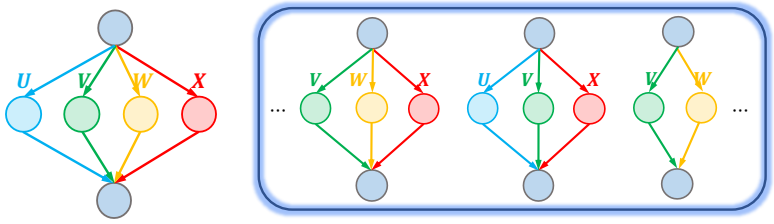
Figure 2.1: A typical flow of Neural Architecture Search.

ization performance. As summarized in (Elsken et al., 2019a), NAS conventionally consists of a search space, a search algorithm, and a performance evaluation. Specifically, the search space of NAS includes all possible candidate architectures that may achieve the best generalization on the target tasks, which usually is manually designed by human experts. The search algorithm then selects the best-performing candidate architecture from this search space based on its evaluated performance via the performance evaluation component in NAS. Figure 2.1 provides a typical flow of NAS.

Given a loss function $\mathcal{L}$ and model parameters $\theta_{\mathcal{A}}$ (or $\theta(\mathcal{A})$) of candidate architecture $\mathcal{A}$, we denote the training and validation loss of this architecture as $\mathcal{L}_{\text{train}}(\theta_{\mathcal{A}}; \mathcal{A})$ and $\mathcal{L}_{\text{val}}(\theta_{\mathcal{A}}; \mathcal{A})$, respectively. NAS can then be formally formulated as a bi-level optimization problem (Liu et al., 2019) below using the validation loss to estimate the true generalization performance of candidate architectures:

$$\begin{aligned} &\min_{\mathcal{A}} \mathcal{L}_{\text{val}}(\theta^*_{\mathcal{A}}; \mathcal{A}) \\ &\text{s.t. } \theta^*_{\mathcal{A}} \triangleq \arg\min_{\theta_{\mathcal{A}}} \mathcal{L}_{\text{train}}(\theta_{\mathcal{A}}; \mathcal{A}) \,. \end{aligned} \tag{2.1}$$

Notably, model training of each candidate architecture is required to



(a) Supernet (b) Candidates

Figure 2.2: The relation between supernet and candidate architectures in one-shot NAS. We denote the first-level, second-level, and third-level nodes as the input, intermediate feature maps, and output for this simple supernet, respectively. Each type of intermediate feature maps is generated by its corresponding operation denoted by the parameters (e.g., $\boldsymbol{X}$) of this operation. Note that candidate architectures (subgraphs of supernet) inherit not only the structure but also the model parameters from the supernet in one-shot NAS, as illustrated here.

evaluate their validation performance in (2.1).

One-shot Neural Architecture Search. As model training of each candidate architectures in the search space is prohibitively costly, one-shot NAS has been widely adopted in the literature (Brock et al., 2018; Liu et al., 2019; Pham et al., 2018) to accelerate the search process of NAS. Particularly, the search space of one-shot NAS is represented as a supernet, and only the model training of this supernet is required. The model parameters of this supernet are then shared among candidate architectures (i.e., the subgraphs of this supernet) to estimate the generalization performance of these candidate architectures in the search space. Interestingly, such estimation is shown to be highly related to the true generalization of candidate architectures in the search space (Bender et al., 2018) and can indeed help to select well-performing architectures with an accelerated search process as validated in (Liu et al., 2019; Pham et al., 2018) empirically. Figure 2.2 illustrates the relation between supernet and candidate architectures in one-shot NAS. Note that throughout this thesis, we may also use one-shot architecture to denote this supernet.

Benchmarks. In recent years, there is a surging interest in developing benchmarks with tabular data to evaluate different NAS algorithms. For example, Ying et al. (2019) are the first to propose a NAS benchmark (i.e., NAS-Bench-101) consisting of the test performances of 423,000 unique architectures on CIFAR-10 (Krizhevsky et al., 2009). These architectures are all independently trained from scratch on the CIFAR-10 training dataset for the same number of training epochs and then are evaluated on the CIFAR-10 test dataset. After that, following the same method in (Ying et al., 2019), Dong and Yang (2020) have constructed a NAS benchmark (i.e., NAS-Bench-201) of 15,625 candidate architectures on various datasets, i.e., CIFAR-10/100 (Krizhevsky et al., 2009) and ImageNet-16-120 (Chrabaszcz et al., 2017), while Zela et al. (2020b) has developed a NAS benchmark (i.e., NAS-Bench-1Shot1) especially for the NAS algorithms using shared parameters from the supernet. These NAS benchmarks significantly reduce the computational cost of evaluating the architectures selected by different NAS algorithms since the performances of these architectures can be simply queried from the benchmarks. Unfortunately, due to the unaffordable computational cost of training all candidate architectures in a large-scale search space (e.g., with $\sim 10^{20}$ unique architectures) as well as a large-scale dataset (e.g., ImageNet (Deng et al., 2009)), these benchmarks typically can only evaluate the performances of NAS algorithms using small-scale search spaces and datasets. So, in the literature, experiments on the large-scale search space (i.e., the DARTS search space (Liu et al., 2019)) and the large-scale dataset (i.e., ImageNet) are usually necessary to further evaluate the practical performance of NAS algorithms. As a result, in this thesis, we mainly follow the convention in the NAS literature to evaluate our NAS algorithms on the aforementioned benchmarks.

2.2 Stochastic Gradient Descent

Nowadays, SGD has become a standard optimization algorithm for the model training of DNNs. Here, we give a brief introduction to SGD. Specifically, given the target function needing to be optimized be in the form of expectation, i.e.,

$$F(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[f(\boldsymbol{w},\boldsymbol{x})]\,, \tag{2.2}$$

to obtain the optimal $\boldsymbol{w}^* = \arg\min F(\boldsymbol{w})$, SGD intends to randomly sample one single data $\boldsymbol{x}_k \sim p(\boldsymbol{x})$ at each iteration to estimate the gradient $\nabla_{\boldsymbol{w}} F(\boldsymbol{w})$ by using $\nabla_{\boldsymbol{w}} f(\boldsymbol{w},\boldsymbol{x}_k)$ and then apply gradient descent onto $\boldsymbol{w}$ with learning rate η_k (detailed in Algorithm 2.1), i.e.,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_k \nabla_{\boldsymbol{w}} f(\boldsymbol{w},\boldsymbol{x}_k)\,. \tag{2.3}$$

Theorem 2.1 provides a guaranteed convergence of SGD. It shows that such convergence rate mainly depends on the Lipschitz constant L, gradient variance bound σ^2, and the choice of learning rate $\{\eta\}_{k=1}^N$. Interestingly, both L and σ^2 are highly related to the neural architectures as justified in Chapter 4. Note that in practice, we usually adopt a mini-batch variant of SGD to enjoy the benefits of parallelization, i.e., we randomly sample a mini-batch of data (i.e., $\{\boldsymbol{x}_k^{(i)}\}_{i=1}^b$) instead of one single data (i.e., $\boldsymbol{x}_k$) to estimate the gradient of $F(\boldsymbol{w})$ at each iteration. In the scenario of *machine learning* and *deep learning*, f, $\boldsymbol{w}$ and $\boldsymbol{x}$ usually denote the composition of model and objective function, model parameters, and data, respectively. Besides, the distribution $p(\boldsymbol{x})$ usually becomes a categorical uniform distribution given a finite number of data in the target dataset.

Algorithm 2.1 Stochastic Gradient Descent (SGD)

1: **Input:** Initialized $\boldsymbol{w}_0$, distribution $p(\boldsymbol{x})$, learning rate $\{\eta_k\}_{k=1}^N$
2: **for** step $k = 0, \ldots, N-1$ **do**
3: Sample $\boldsymbol{x}_k \sim p(\boldsymbol{x})$
4: Update $\boldsymbol{w}$ via $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta_k \nabla_{\boldsymbol{w}} f(\boldsymbol{w}_k, \boldsymbol{x}_k)$

Theorem 2.1. *(Ghadimi and Lan, 2013) Let F be a L-smooth non-convex function, and let F^* be its minimal. Given repeated, independent accesses to stochastic gradients with variance bound σ^2 for $F(\boldsymbol{w})$, SGD with initial $\boldsymbol{w}_0$, total iterations $N > 0$ and learning rate $\eta_k < 1/L$ achieves the following convergence by randomly choosing $\boldsymbol{w}_k$ as the final output $\boldsymbol{w}_R$ with probability η_k/H where $H = \sum_{k=1}^N \eta_k$:*

$$\mathbb{E}[\left\|\nabla F(\boldsymbol{w}_R)\right\|^2] \leq \frac{2(F(\boldsymbol{w}_0) - F^*)}{H} + \frac{L\sigma^2}{H} \sum_{k=1}^{N} \eta_k^2$$

2.3 Stein Variational Gradient Descent

Stein Variational Gradient Descent (SVGD) (Liu and Wang, 2016) is a variational inference algorithm that approximates a target distribution $p(\boldsymbol{x})$ with a simpler density $q^*(\boldsymbol{x})$ in a predefined set $\mathcal{Q}$, which is selected by minimizing the *Kullback-Leibler* (KL) divergence between these two densities:

$$q^* = \arg\min_{q \in \mathcal{Q}} \{\mathrm{KL}(q \| p) \triangleq \mathbb{E}_q \left[\log\left(q(\boldsymbol{x})/p(\boldsymbol{x})\right)\right]\} . \tag{2.4}$$

Specifically, SVGD represents $q^*(\boldsymbol{x})$ with a set of particles $\{\boldsymbol{x}_i\}_{i=1}^n$, which are firstly randomly initialized and then iteratively updated with updates $\boldsymbol{\phi}^*(\boldsymbol{x}_i)$ and a step size ϵ:

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon \boldsymbol{\phi}^*(\boldsymbol{x}_i), \quad \forall i \in \{1, \cdots, n\} . \tag{2.5}$$

Let $q_{[\epsilon\phi]}$ denote the distribution of updated particles $\boldsymbol{x}' = \boldsymbol{x} + \epsilon\boldsymbol{\phi}(\boldsymbol{x})$. Let $\mathbb{F}$ denote the unit ball of a vector-valued reproducing kernel Hilbert space (RKHS) $\mathcal{H} \triangleq \mathcal{H}_0 \times \cdots \mathcal{H}_0$ where $\mathcal{H}_0$ is an RKHS formed by scalar-valued functions associated with a positive definite kernel $k(\boldsymbol{x}, \boldsymbol{x}')$. The work of (Liu and Wang, 2016) has shown that (2.5) can be viewed as functional gradient descent in the RKHS $\mathcal{H}$ and the optimal $\boldsymbol{\phi}^*$ in (2.5) can be obtained by solving the following problem

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi} \in \mathbb{F}} \left\{ -\frac{d}{d\epsilon} \mathrm{KL}(q_{[\epsilon\boldsymbol{\phi}]} \| p) \Big|_{\epsilon=0} \right\}, \tag{2.6}$$

which yields a closed-form solution:

$$\boldsymbol{\phi}^*(\cdot) = \mathbb{E}_{\boldsymbol{x} \sim q}[k(\boldsymbol{x}, \cdot) \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} k(\boldsymbol{x}, \cdot)] \,. \tag{2.7}$$

In practice, Liu and Wang (2016) have approximated the expectation in this close-form solution with the empirical mean of particles: $\boldsymbol{\phi}^*(\boldsymbol{x}_i) \approx \widehat{\boldsymbol{\phi}}^*(\boldsymbol{x}_i)$, where $\widehat{\boldsymbol{\phi}}^*(\boldsymbol{x}_i)$ is defined as

$$\widehat{\boldsymbol{\phi}}^*(\boldsymbol{x}_i) \triangleq \frac{1}{n} \sum_{j=1}^{n} k(\boldsymbol{x}_j, \boldsymbol{x}_i) \nabla_{\boldsymbol{x}_j} \log p(\boldsymbol{x}_j) + \nabla_{\boldsymbol{x}_j} k(\boldsymbol{x}_j, \boldsymbol{x}_i) \,. \tag{2.8}$$

As revealed in (Liu and Wang, 2016), the two terms in the aforementioned closed-form solution take different effects: The first term with $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ favors particles with higher probability density, while the second term pushes the particles away from each other to encourage diversity.

2.4 Neural Tangent Kernel

Let a dataset $(\mathcal{X}, \mathcal{Y})$ denote a pair comprising a set $\mathcal{X}$ of m n_0-dimensional vectors of input features and a vector $\mathcal{Y} \in \mathbb{R}^{mn \times 1}$ concatenating the m

n-dimensional vectors of corresponding output values, respectively. Let a DNN be parameterized by $\boldsymbol{\theta}_t \in \mathbb{R}^p$ at time t and output a vector $\boldsymbol{f}(\mathcal{X};\boldsymbol{\theta}_t) \in \mathbb{R}^{mn\times 1}$ (abbreviated to $\boldsymbol{f}_t$) of the predicted values of $\mathcal{Y}$. Jacot et al. (2018) have revealed that the training dynamics of DNNs with gradient descent can be characterized by an NTK. Formally, define the NTK $\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X}) \in \mathbb{R}^{mn\times mn}$ (abbreviated to $\boldsymbol{\Theta}_t$) as

$$\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X}) \triangleq \nabla_{\boldsymbol{\theta}_t}\boldsymbol{f}(\mathcal{X};\boldsymbol{\theta}_t)\,\nabla_{\boldsymbol{\theta}_t}\boldsymbol{f}(\mathcal{X};\boldsymbol{\theta}_t)^\top . \tag{2.9}$$

Given a loss function $\mathcal{L}_t$ at time t and a learning rate η, the training dynamics of the DNN can then be characterized as

$$\nabla_t \boldsymbol{f}_t = -\eta\,\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X})\,\nabla_{\boldsymbol{f}_t}\mathcal{L}_t, \quad \nabla_t \mathcal{L}_t = -\eta\,\nabla_{\boldsymbol{f}_t}\mathcal{L}_t^\top\,\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X})\,\nabla_{\boldsymbol{f}_t}\mathcal{L}_t . \tag{2.10}$$

Interestingly, as proven in (Jacot et al., 2018), the NTK stays asymptotically constant during the course of training as the width of DNNs goes to infinity. NTK at initialization (i.e., $\boldsymbol{\Theta}_0$) can thus characterize the training dynamics and also the performance of infinite-width DNNs.

Lee et al. (2019a) have further revealed that, for DNNs with over-parameterization, the aforementioned training dynamics can be governed by their first-order Taylor expansion (or linearization) at initialization. In particular, define

$$\boldsymbol{f}^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}_t) \triangleq \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}_0}\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) \tag{2.11}$$

for all $\boldsymbol{x} \in \mathcal{X}$. Then, $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_t)$ and $\boldsymbol{f}^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}_t)$ share similar training dynamics over time, as described formally in Appendix C.1.2. Besides, following the definition of NTK in (2.9), this linearization $\boldsymbol{f}^{\text{lin}}$ achieves a constant NTK over time.

Given the mean squared error (MSE) loss defined as $\mathcal{L}_t \triangleq m^{-1} \left\| \mathcal{Y} - \boldsymbol{f}(\mathcal{X}; \boldsymbol{\theta}_t) \right\|_2^2$ and the constant NTK $\boldsymbol{\Theta}_t = \boldsymbol{\Theta}_0$, the loss dynamics in (2.10) above can be analyzed in a closed form while applying gradient descent with learning rate η (Arora et al., 2019b):

$$\mathcal{L}_t = m^{-1} \sum_{i=1}^{mn} (1 - \eta \lambda_i)^{2t} (\boldsymbol{u}_i^\top \mathcal{Y})^2 \,, \tag{2.12}$$

where $\boldsymbol{\Theta}_0 = \sum_{i=1}^{mn} \lambda_i(\boldsymbol{\Theta}_0) \boldsymbol{u}_i \boldsymbol{u}_i^\top$, and $\lambda_i(\boldsymbol{\Theta}_0)$ and $\boldsymbol{u}_i$ denote the i-th largest eigenvalue and the corresponding eigenvector of $\boldsymbol{\Theta}_0$, respectively.

Chapter 3

Related Works

In this chapter, we provide a review of related works in NAS to elucidate the position of our contributions within the literature.

3.1 Evolution of Training-based NAS

NAS has received increasing attention in recent years due to its outstanding performance and the demand for Automated Machine Learning (AutoML). There are three major components in NAS as summarized by (Elsken et al., 2019b), namely search space, search policy (or strategy, algorithm), and performance evaluation (or estimation). To define the search space, the prior knowledge extracted from expert-designed architectures is typically exploited. As for the search policy, different algorithms are proposed to improve the effectiveness (Cai et al., 2019a; Real et al., 2019a; Tan et al., 2019a; Zoph et al., 2018) of NAS, which we introduce in the following paragraphs in detail.

NASNet. Zoph and Le (2017) and Zoph et al. (2018) are the first batch to introduce neural architecture search to automate the design of neural networks. Specifically, they have proposed to use reinforcement (RL)

algorithms to select best-performing architectures, in which a recurrent neural network (RNN) is applied as the controller to iteratively sample high-potential architectures, and then the accuracies of these fully trained architectures are employed to update the RNN in order to get a better controller. While Zoph and Le (2017) propose to search a whole neural architecture from scratch, Zoph et al. (2018) only searches for two different cells (i.e., normal and reduce cells) where a complete neural architecture is stacked by a number of these two types of cells. Thanks to the decomposition of neural architectures as cells, Zoph et al. (2018) have successfully scaled their final selected architectures (namely NASNet) to other benchmarks tasks. Though empirical results show that NASNet is able to achieve SOTA performances, its prohibitive search cost derived from the model training of each sampled architecture is unaffordable for many NAS practitioners.

AmoebaNet. After NASNet, Real et al. (2019a) have proposed to employ an evolutionary algorithm to conduct NAS using the same search space as NASNet. Specifically, Real et al. (2019a) have developed an aging evolution algorithm based on standard tournament selection evolutionary algorithm by novelly introducing age to each candidate and then preferring young candidates. Such a NAS algorithm is usually referred to AmoebaNet algorithm. Empirical experiments show that the AmoebaNet algorithm can usually enjoy a faster search process than RL-based NASNet and also NAS algorithm using random search, i.e., requiring a smaller search cost to achieve the same performance. Meanwhile, AmoebaNet usually will achieve competitive performance compared with NASNet in practice. Similar to NASNet, each sampled candidate architecture during the search process of the AmoebaNet algorithm is also required to be

trained from scratch independently.

ProxylessNAS. Instead of searching on a proxy task and then transferring the selected architectures from this proxy task to the target task, Cai et al. (2019a) have proposed to directly search for the target tasks as well as the target hardware platforms using the ProxylessNAS algorithm. Particularly, following that of (Liu et al., 2019), Cai et al. (2019a) have represented the search space as a directed acyclic graph (DAG) and use binarized architecture parameters to indicate each candidate architecture in the search space. Then, Cai et al. (2019a) have employed a REINFORCE-based approach to optimize these binarized architecture parameters on the validation dataset of the target task in order to select the best-performing architecture from the search space. Empirical results show that ProxylessNAS has achieved considerably improved performance on various datasets compared with NASNet and AmoebaNet. Moreover, since ProxylessNAS only needs to train the supernet during their search process, it will also incur a significantly reduced cost than NASNet and AmoebaNet. However, the cost of direct search on large-scale datasets in ProxylessNAS is still unsatisfactory in practice.

In practice, scaling the aforementioned NAS algorithms to large datasets is notoriously hard. Recently, attention has thus been shifted to improving the search efficiency of NAS without sacrificing the generalization performance of its selected architectures. In particular, a supernet (or one-shot architecture) is firstly introduced by Pham et al. (2018) to share model parameters among candidate architectures, thereby reducing the cost of model training substantially. Recent works (Chen et al., 2019; Dong and Yang, 2019b; Liu et al., 2019; Xie et al., 2019b; Chen and Hsieh, 2020; Chu et al., 2020) along this line have further formulated NAS as

a continuous and differentiable optimization problem to yield efficient gradient-based solutions. We introduce certain typical one-shot NAS algorithms in detail in the following paragraphs.

ENAS. Considering the expensive search cost of the aforementioned NAS algorithms, Pham et al. (2018) has framed the search space of NAS into a supernet (or a computational graph) and then shared model parameters among candidate architectures (i.e., the sub-graph of this supernet) in such a search space. In particular, Pham et al. (2018) has developed a search process consisting of two interleaving phases: In the first phase, candidate architectures are sampled from the search space by a similar RNN controller as (Zoph et al., 2018), and then their shared model parameters with supernet are trained on the training dataset for only a few steps. In the second phase, the aforementioned RNN controller is updated using the policy gradient derived from the validation performance of those sampled architectures. By developing such a weight-sharing mechanism, only the model training of this supernet is required to search for best-performing architectures, which therefore will be more efficient compared with the costly model training of multiple architectures in NASNet and AmoebaNet. Empirical results have also validated the improved search efficiency over NASNet and AmoebaNet achieved by (Pham et al., 2018).

DARTS. Most of the aforementioned NAS algorithms typically use reinforcement algorithms or evolutionary algorithms to search architectures in a discrete and non-differentiable search space. In practice, gradient-based optimization algorithms typically can achieve a faster convergence than reinforcement algorithms or evolutionary algorithms. As a result, to achieve a further improved search efficiency for NAS, Liu et al. (2019)

have developed the DARTS algorithm to search architectures in a differentiable search space using gradients. Particularly, (Liu et al., 2019) have novelly relaxed the discrete search space as a continuous one by using a weighted combination (parameterized by architecture parameters) for all possible operations and then proposed to use a bi-level gradient-based optimization algorithm to update both the model parameters and the architecture parameters. Note that operations are used to make up a complete architecture. Besides, due to such a relaxation, the search space is also framed as a supernet and the model parameters of candidate architectures are also shared with each other. Extensive experiments have shown that DARTS is capable of scaling down the search cost of NASNet and AmoebaNet to several orders of magnitude while still enjoying competitive test performances.

SNAS & GDAS. Though DARTS has achieved impressive results in practice, there usually exists a large gap between the performance of candidate architectures during the search process and after the search process because of the discrepancy between continuous (during the search process) and binary (after the search process) architecture parameters. In light of this, Xie et al. (2019b) and Dong and Yang (2019b) have introduced probability distribution to binary architecture parameters using Gumbel softmax (Maddison et al., 2017; Jang et al., 2017) compared with the continuous ones in (Liu et al., 2019). That is, architecture parameters will remain discrete during their search process. As a result, the gap between the performance of candidate architectures during the search process and after the search process will disappear, which has also been validated by their empirical results.

SDARTS. Moreover, DARTS also has the problem of achieving unstable search results in practice. To this end, Chen and Hsieh (2020) have visualized the validation loss landscape and observed that the precipitous landscape is the main reason for why there exists a performance drop when selecting the final architecture in DARTS. Motivated by this, Chen and Hsieh (2020) have developed SDARTS algorithm to overcome the instability and the lack of generalizability of DARTS by smoothing the loss landscape with random or adversarial noise. Their Analysis has shown that these two smoothing methods can theoretically reduce the spectral norm of Hessian (which serves as a quantitative way to measure the smoothness of loss landscape) and, therefore can lead to a benign loss landscape. Extensive empirical results have also shown the superiority of SDARTS over DARTS by achieving stable and improved search performances while incurring comparable search costs.

3.2 Evaluating Training-based NAS

In the literature, no effort has been devoted to understanding the best architectures selected by various popular NAS approaches before our work in Chapter 4. Instead, recent works attempt to evaluate NAS algorithms by comparing them with random search. Specifically, the generalization performance of architectures selected with existing NAS algorithms is compared with the architectures selected with random search in (Li and Talwalkar, 2019a) and (Sciuto et al., 2019). Interestingly, random search is shown to be able to find architectures with comparable or even better generalization performance. Particularly, the ineffectiveness of certain NAS algorithms, i.e., (Pham et al., 2018), could be the consequence of the weight sharing mechanism during the search process as revealed in

(Sciuto et al., 2019). While these evaluations help understand the general disadvantages of NAS algorithms, what kind of architectures the NAS algorithms are selecting and why they select these specific architectures are still not well understood, which is exactly what we will answer in Chapter 4.

3.3 Neural Network Ensembles

In practice, neural network ensembles have been widely applied to improve the performance of a single neural network in different applications (Dietterich, 2000). Over the years, a number of methods have been proposed to construct such neural network ensembles, which we introduce in the following paragraphs in detail.

Monte Carlo Dropout. As Dropout (Srivastava et al., 2014) has been widely applied to avoid model over-fitting, Gal and Ghahramani (2016) have novelly reformulated it as a Bayesian Approximation to interpret its compelling performance. Based on such an interpretation, Gal and Ghahramani (2016) have proposed to use Monte Carlo Dropout to obtain neural network ensembles (for model uncertainty) by randomly dropping out neurons in neural networks at test time. Empirical results have shown that Monte Carlo Dropout can indeed provide reliable model uncertainty for neural networks.

Deep Ensemble. After Monte Carlo Dropout, *deep ensembles* (DeepEns) (Lakshminarayanan et al., 2017) then propose to adopt neural networks trained with different random initializations to construct neural network ensembles. Compared with Monte Carlo Dropout, DeepEns has been shown to be able to achieve improved predictive performances and un-

certainty on various tasks. However, DeepEns is more computationally expensive than Monte Carlo Dropout because DeepEns needs to train a neural network from scratch for multiple times while Monte Carlo Dropout only needs to train a neural network once.

Snapshot Ensemble. Considering that DeepEns is too computationally expensive, Huang et al. (2017a) have proposed a novel neural network ensemble that only requires one-time model training, namely snapshot ensemble. Specifically, Huang et al. (2017a) have employed a cyclic learning rate schedule to train a single neural network and then applied the checkpoints obtained during the model training of this neural network to construct neural network ensembles (Huang et al., 2017a). As a result, snapshot ensemble has successfully avoided the multiple model training required by DeepEns and thus is more computationally efficient than DeepEns. Empirical results also show that snapshot ensemble is capable of achieving compelling performance in practice.

Neural Ensemble Search. Though the aforementioned ensemble methods are already able to achieve competitive performance in practice, each candidate in these ensembles is usually randomly selected and therefore can usually be far away from the optimal candidates to realize the best-performing ensemble. Moreover, the aforementioned ensemble methods mainly rely on a single neural network to construct ensembles. So, the diversity of the candidates in these ensembles is usually highly restricted, which is known to be harmful to the final ensemble performance according to (Zhou, 2012). To this end, more recently, Zaidi et al. (2020) have introduced *neural ensemble search* (NES) into the NAS area to build well-performing neural network ensembles by selecting diverse architectures from the NAS search space, which has achieved competitive or

even improved performance compared with the aforementioned ensemble methods. This therefore implies the superiority of neural ensemble search over other ensemble methods. Unfortunately, the algorithm presented in (Zaidi et al., 2020) is shown to be prohibitively costly due to the requirement of training multiple neural networks from scratch. Moreover, the performance of the algorithm in (Zaidi et al., 2020) is severely restricted by another requirement of sampling only a pool of architectures from the search space for the ensemble search. So, our work in Chapter 5 intends to tackle these two problems.

3.4 Training-free NAS

Though one-shot NAS algorithms have achieved considerable improvement in search efficiency, the model training of the one-shot architecture (or supernet) is still required. More recently, a number of algorithms have been proposed to estimate the performance of candidate architectures without model training and therefore can avoid the model training of neural networks completely during the search process of NAS. We introduce them in detail in the following paragraphs.

NASWOT. Mellor et al. (2020a) are the first to explore the training-free in the literature. Specifically, for the intermediate feature maps of each layer after the ReLU activation function that are induced by an input, they use a binary indicator to examine whether these intermediate feature maps are positive values or not. After that, they heuristically employ the correlation among the indicator variables induced by different data points using only neural networks at initialization to estimate the performance of candidate architectures in the search space. Finally, they propose to select the architecture that is capable of achieving the best

correlation in the search space as their final selected architecture, namely NASWOT algorithm. Empirical results show that such a method can indeed help to find architectures achieving competitive performances with only a fraction of the search cost in training-based NAS algorithms on NAS-Bench-101/201. Unfortunately, Mellor et al. (2020a) fail to provide theoretical justification for their training-free NAS algorithm. Meanwhile, they also fail to provide the performance of NASWOT in large-scale search space (i.e., the DARTS search space) and dataset (i.e., ImageNet).

NNGP-NAS. Recently, de G. Matthews et al. (2018) have shown that neural networks that are initialized using Gaussian distributions correspond to a special Gaussian process, namely Neural Network Gaussian Process (NNGP). Inspired by this, Park et al. (2020) have proposed to employ the posterior of this NNGP (obtained by Bayesian inference on the training dataset of the target task) to estimate the performance of each candidate architecture on the test dataset of the target task. Though Park et al. (2020) have conducted plenty of empirical studies to validate the feasibility of such a training-free NAS method, they did not provide the performances of the final selected architectures by their NNGP-NAS. As a result, it is hard to evaluate the effectiveness of their NNGP-NAS. Meanwhile, to obtain the estimated test performance of each candidate architecture in the search space, a $N_{\text{train}} \times N_{\text{train}}$ kernel matrix needs to be inverted, which is yet computationally costly. Moreover, though empirical results show that NNGP-NAS can be used to approximate the true performances of candidate architectures, there is no theoretical result to show how accurate such an approximation can be as well as how to further improve the quality of such an approximation.

Zero-Cost Proxies. Inspired by the aforementioned works, Abdelfattah et al. (2021) have proposed to use a number of zero-cost proxies to estimate the test performances of architectures. Specifically, they have introduced gradient norm, SNIP (Lee et al., 2019b), GraSP (Wang et al., 2020a), fisher (Turner et al., 2020) and Synflow (Tanaka et al., 2020) to conduct training-free NAS, most of which are originally developed for training-free network pruning. They firstly investigated the quality of these zero-cost proxies to estimate the true performances of candidate architectures by computing the correlation between these zero-cost proxies and the true performances of the architectures in the search space. Empirical results show that most of them can usually provide useful information about the true performances of architectures. As a result, they further proposed to use the architectures selected by these zero-cost proxies as good initialization (i.e., warm-up) for training-based NAS algorithms, which have shown to be more efficient than standard NAS algorithms. Unfortunately, they can only achieve very limited improvement over standard training-based NAS algorithms in practice. Meanwhile, these zero-cost proxies themselves usually fail to select architectures that can achieve competitive performance as those selected by training-based NAS algorithms. Therefore, *how to further improve the quality of these zero-cost proxies* remains a mystery.

TE-NAS. Recently, Jacot et al. (2018) have developed a theory of Neural Tangent Kernel (NTK), aiming to characterize the training dynamics of neural networks theoretically. Meanwhile, in the literature, the number of linear regions for neural networks has been applied to measure the expressivity of neural networks (Raghu et al., 2017). Inspired by these two types of studies on neural networks, Chen et al. (2021) have intuitively

proposed to use not only the condition number of the NTK matrix but also the number of linear regions to select architectures with a good trade-off between its trainability and expressivity, which is named TE-NAS algorithm. Surprisingly, TE-NAS can achieve competitive performance even compared with training-based NAS algorithms while the search cost has been considerably reduced on various benchmarks, e.g., NAS-Bench-201 and the DARTS search space on ImageNet. Unfortunately, TE-NAS is mainly based on intuitive inspirations and thus fails to provide theoretical guarantees to its compelling performance. Moreover, the trade-off in TE-NAS is usually manually decided, which therefore lacks theoretical insights on how to find the optimal trade-off for different NAS tasks.

KNAS. Similarly, following the theory of NTK, Xu et al. (2021) have applied the mean of the elements in a so-called Gram matrix of the gradients with respect to the model parameters to help them estimate the true performance of architectures quickly. Then, Xu et al. (2021) have proposed to firstly maintain a pool of top-k architectures that can achieve the highest scores based on such a performance approximation and then select the architecture achieving the best validation performance after a small number of model training from this pool. Such a method is named KNAS by them. Though KNAS can achieve improved search efficiency compared with training-based NAS algorithms, such an improvement is considerably smaller than other training-free NAS baselines, e.g., NASWOT and TE-NAS. Moreover, KNAS fails to achieve competitive performance even compared with TE-NAS and the empirical results on large-scale benchmarks have not been provided to further validate the effectiveness of KNAS.

Above all, considering the shortcomings of the aforementioned training-free NAS algorithms, we propose a novel training-free NAS algorithm named NASI in Chapter 6 to provide theoretical guarantees for its performance as well as transferability and to achieve SOTA performance on the widely applied benchmarks in the NAS area. After that, we then present a unified analysis for those gradient-based training-free metrics to theoretically guarantee the performance and transferability of the training-free NAS algorithms using these metrics in Chapter 7. Meanwhile, based on our unified analysis, we further develop a novel NAS algorithm named HNAS to further improve the performance of training-free NAS while maintaining its search efficiency Chapter 7.

Chapter 4

Understanding Architectures Learnt by Cell-based Neural Architecture Search

This chapter is based on the following work published at ICLR 2020: Shu, Y., Wang, W., & Cai, S. (2020). Understanding architectures learnt by cell-based neural architecture search. In *Proc. ICLR-20*.

4.1 Introduction

Various neural network architectures (He et al., 2016; Huang et al., 2017b; Krizhevsky et al., 2012; Simonyan and Zisserman, 2015) have been devised over the past decades, achieving superhuman performance for a wide range of tasks. Designing these neural networks typically takes substantial efforts from domain experts by trial and error. Recently, there is a growing interest in *neural architecture search* (NAS), which automatically searches for high-performance architectures for the given task. The searched NAS architectures (Akimoto et al., 2019; Cai et al., 2019a; Liu

et al., 2019; Luo et al., 2018b; Nayman et al., 2019; Pham et al., 2018; Real et al., 2019a; Xie et al., 2019b; Zoph et al., 2018) have outperformed best expert-designed architectures on many computer vision and natural language processing tasks.

Mainstream NAS algorithms typically search for the connection topology and transforming operation accompanying each connection from a predefined search space. In the literature, tremendous efforts have been exerted to develop efficient and effective NAS algorithms (Akimoto et al., 2019; Liu et al., 2019; Luo et al., 2018b; Nayman et al., 2019; Xie et al., 2019b). However, less attention has been paid to these searched architectures for further insight. To our best knowledge, there is no related work in the literature examining whether these NAS architectures share any pattern, and how the pattern may impact the architecture search if there exists the pattern. These questions are fundamental to understand and improve existing NAS algorithms. In this work, we endeavour to address these questions by examining the popular NAS architectures[1].

The recent work (Xie et al., 2019a) shows that the architectures with random connection topologies can achieve competitive performance on various tasks compared with expert-designed architectures. Inspired by this result, we examine the connection topologies of the architectures generated by popular NAS algorithms. In particular, we find a connection pattern of the popular NAS architectures. These architectures tend to favor wide and shallow cells, where the majority of intermediate nodes are directly connected to the input nodes.

To appreciate this particular connection pattern, we first visualize the training process of the popular NAS architectures and their ran-

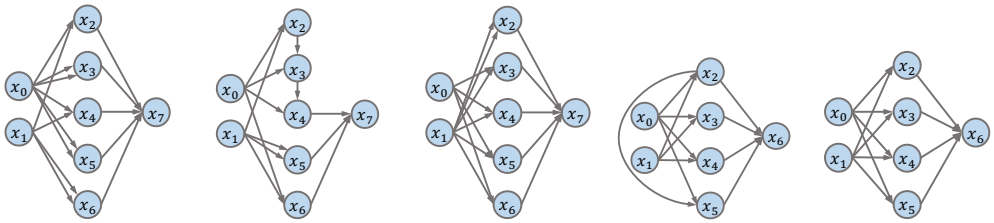
[1]The popular NAS architectures refer to the best architectures generated by state-of-the-art (SOTA) NAS algorithms throughout the work. Notably, we research on the cell-based NAS algorithms and architectures.

domly connected variants. Fast and stable convergence is observed for the architectures with wide and shallow cells. We further empirically and theoretically show that the architectures with wider and shallower cells consistently enjoy a smoother loss landscape and smaller gradient variance than their random variants, which helps explain their better convergence and consequently the selection of these NAS architectures during the architecture search.

We finally evaluate the generalization performance of the popular NAS architectures and their randomly connected variants. We find that the architectures with wide and shallow cells may not generalize better than other candidate architectures despite their faster convergence. We therefore believe that rethinking NAS from the perspective of the true generalization performance rather than the convergence of candidate architectures should potentially help generate better architectures.

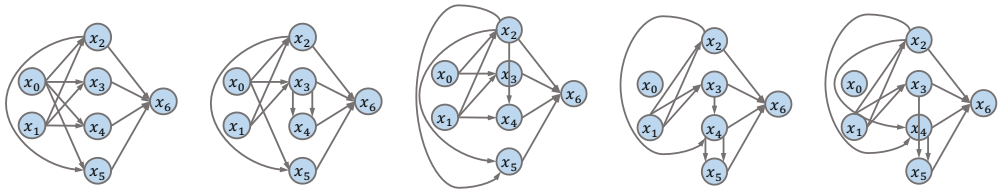
4.2 The Connection Pattern of Popular NAS Cells

Mainstream NAS algorithms (Liu et al., 2019; Luo et al., 2018b; Pham et al., 2018; Real et al., 2019a; Xie et al., 2019b; Zoph et al., 2018) typically search for the cell structure, including the connection topology and the corresponding operation (transformation) coupling each connection. The generated cell is then replicated to construct the entire neural network. We therefore mainly investigate these cell-based NAS architectures. In this section, we first introduce the commonly adopted cell representation, which is useful to understand the connection and computation in a cell space. We then sketch the connection topologies of popular cell-based NAS architectures to investigate their connection patterns. By comparison, we show that there is a common connection pattern among the cells



(a) NASNet, $5c$, 2 (b) AmoebaNet, $4c$, 4 (c) ENAS, $5c$, 2 (d) DARTS, $3.5c$, 3 (e) SNAS, $4c$, 2

Figure 4.1: Cell topologies of popular NAS architectures. Each sub-figure has three sets of nodes from left to right, i.e., the input nodes, intermediate nodes, and output node. The arrows (i.e., operations of the cell) represent the direction of information flow. The caption of each sub-figure reports the name of the architecture, width and depth of a cell following our definition. The width of a cell is computed with the assumption that all intermediate nodes share the same width c.



(a) C^{darts}, $3.5c$, 2 (b) C_1^{darts}, $2.5c$, 3 (c) C_2^{darts}, $2.5c$, 3 (d) C_3^{darts}, $2c$, 4 (e) C_4^{darts}, $2c$, 4

Figure 4.2: Topologies of DARTS (Liu et al., 2019) cell (leftmost) and its variants with random connections. The cell depth is increasing and width decreasing from left to right. In particular, the original DARTS cell C^{darts} is widest and shallowest among these cells.

learned by different NAS algorithms; particularly, these cells tend to be wide and shallow.

4.2.1 Cell Representation

Following DARTS (Liu et al., 2019), we represent the cell topology as a directed acyclic graph (DAG) consisting of N nodes, including M input nodes, one output node and $(N-M-1)$ intermediate nodes. Each node forms a latent representation of the input instance. The input nodes consist of the outputs from M preceding cells. And the output node aggregates (e.g., concatenate) the representations from all intermediate

nodes. Each intermediate node is connected to M proceeding nodes in the same cell. Each connection transforms the representation from one node via an operation from a predefined operation set, e.g., 3×3 convolution, 3×3 max pooling, etc. The target of NAS algorithm is to search for the best M source nodes for each intermediate node and the best operation for each of the connections between nodes. In the literature, the searched cell is then replicated by L times to build the entire neural network architecture[2].

We abuse the notation C to denote a cell and also the architecture built with the specific cell in the following sections. Besides, we shall use C^A to denote the best architecture (or cell) searched with the NAS algorithm A (e.g., DARTS (Liu et al., 2019), ENAS (Pham et al., 2018)). Details on how to build the architecture with given cells are provided in Appendix A.1.3.

4.2.2 The Common Connection Pattern

Recently, it has been shown that neural networks constructed by cells with random connection patterns can achieve compelling performance on multiple tasks in (Xie et al., 2019a). Taking this a step further, we wonder whether cells generated from popular NAS algorithms share any connection patterns, which may explain why these cells are chosen during the architecture search. To investigate the connection patterns, we sketch the topologies of the popular NAS cells with detailed operations omitted.

Figure 4.1 illustrates topologies of 5 popular NAS cells[3]. To examine the connection pattern formally, we introduce the concept of 'depth'

[2]We omit reduction cell here for brevity, which is used for dimension reduction in NAS.

[3]We only visualize normal cells since the number of normal cells is significantly larger than the reduction cells in popular NAS architectures.

and 'width' for a cell. The depth of a cell is defined as the number of connections along the longest path from input nodes to the output node. The width of a cell is defined as the total width of the intermediate nodes that are connected to the input nodes. In particular, if some intermediate nodes are only partially connected to input nodes (i.e., have connections to other intermediate nodes), their width is reduced by the percentage of the number of connections to intermediate nodes over all connections. The width of a node is the number of channels for convolution operations; and the width is the dimension of the features for linear operations. Supposing that the width of each intermediate node is c, as shown in Figure 4.1, the width and depth of the DARTS (Liu et al., 2019) cell are $3.5c$ and 3 respectively, and the width and depth of the AmoebaNet (Real et al., 2019a) cell are $4c$ and 4 correspondingly.

Following the above definitions, the smallest depth and largest width for a cell with $N = 7$ and $M = 2$ are 2 and $4c$ respectively. Similarly, for a cell with $N = 8$ and $M = 2$, the smallest depth and largest width are 2 and $5c$ respectively. In Figure 4.1, we can observe that cells from popular NAS architectures tend to be the widest and shallowest ones (with width close to $4c/5c$ and depth close to 2) among all candidate cells in the same search space. Regarding this as the common connection pattern, we have the following observation:

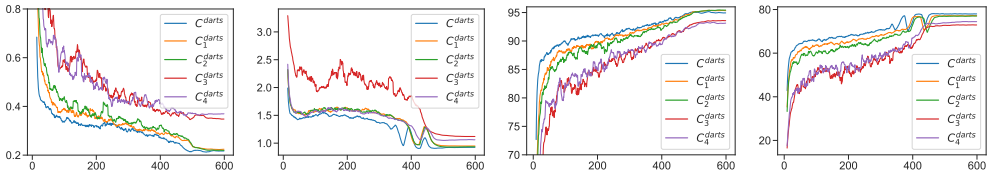
Observation 1 (The Common Connection Pattern)**.** *NAS architectures generated by popular NAS algorithms tend to have the widest and shallowest cells among all candidate cells in the same search space.*

4.3 The Impacts of Cell Width and Depth on Optimization

Given that popular NAS cells share the common connection pattern, we then explore the impact of this common connection pattern from the optimization perspective to answer the question: *why the wide and shallow cells are selected during the architecture search?* We sample and train variants of popular NAS architectures with random connections. Comparing randomly connected variants with the popular NAS architectures, we find that architectures with wider and shallower cells indeed converge faster so that they are selected by NAS algorithms (Section 4.3.1). To understand why the wider and shallower cell contributes to faster convergence, we further investigate the loss landscape and gradient variance of popular NAS architectures and their variants via both empirical experiments (Section 4.3.2) and theoretical analysis (Section 4.3.3).

4.3.1 Convergence

Popular NAS algorithms typically evaluate the performance of a candidate architecture prematurely before the convergence of its model parameters during the search process. For instance, DARTS (Liu et al., 2019), SNAS (Xie et al., 2019b) and ENAS (Pham et al., 2018) optimize hyper-parameters of architectures and model parameters concurrently. The amortized training time of each candidate architecture is insufficient and therefore far from the requirement for the full convergence. Likewise, AmoebaNet (Real et al., 2019a) evaluates the performance of candidate architectures with the training of only a few epochs. In other words, these candidate architectures are not evaluated based on their generalization performance at convergence. As a result, architectures with faster



(a) CIFAR-10 Loss (b) CIFAR-100 Loss (c) CIFAR-10 Accuracy (d) CIFAR-100 Accuracy

Figure 4.3: Test loss and test accuracy (%) curves of DARTS and its randomly connected variants on CIFAR-10 and CIFAR-100 during training. The default learning rate is 0.025.

convergence rates are more likely to be selected by existing NAS algorithms because they can obtain better evaluation performance given the same training budget. We therefore hypothesize that the popular NAS architectures may converge faster than other candidate architectures, which largely contributes to the selection of these architectures during the search.

To support the hypothesis above, we compare the convergence of original NAS architectures and their variants with random connections via empirical studies. We first sample variants of popular NAS cells following the sampling method in Appendix A.1.2. Then, we train both original NAS architectures and their random variants on CIFAR-10 and CIFAR-100 following the training details in Appendix A.1.3. During training, we evaluate the testing loss and accuracy of these architectures. Since the convergence is dependent on optimization settings, we also evaluate the convergence performance under different learning rates.

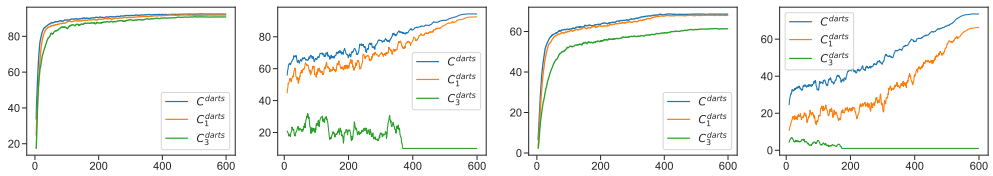
Take DARTS (Liu et al., 2019) for example, Figure 4.2 shows the connection topology of the original DARTS cell and its random variants. Figure 4.3 reports the test loss and accuracy curves of these architectures during training. As illustrated in Figure 4.3, the original cell C^{darts}, known as the widest and shallowest cell, has the fastest and most stable convergence compared with its variants. Further, as the width of a cell

increases and the depth decreases (i.e., from C_4 to C_1), the convergence becomes faster. The results of other popular NAS architectures and their randomly connected variants are reported in Sec. 4.5.2.

Figure 4.4 further validates the difference of convergence under different learning rates. The original cell C^{darts} enjoys the fastest and the most stable convergence among these cells under various learning rates. The difference in terms of convergence rate and stability is more obvious between C^{darts} and its variants with a larger learning rate as shown in Figure 4.4. Interestingly, C_3^{darts} completely fails to converge on both CIFAR-10 and CIFAR-100 with a larger learning rate of 0.25. While there is a minor difference among these cells with a lower learning rate of 0.0025, we still find that there is a decreasing performance of convergence (i.e., convergence rate and stability) from C^{darts}, C_1^{darts} to C_3^{darts}. Overall, the observations are consistent with the results in Figure 4.3.

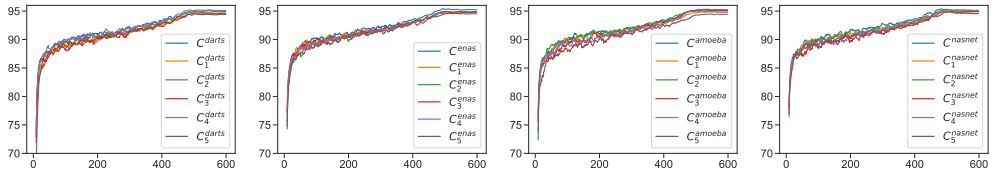
We have also compared the convergence of popular NAS architectures and their random variants of different operations. Similarly, we sample and train the random variants of operations for popular NAS architectures following the details in Appendix A.1.2 and Appendix A.1.3. Figure 4.5 illustrates the convergence of these architectures. Surprisingly, with the same connection topologies as the popular NAS cells but different operations, all random variants achieve nearly the same convergence as these popular NAS architectures. Consistent results can be found in Figure 4.13 of Sec. 4.5.2. We therefore believe that the types of operations have limited impacts on the convergence of NAS architectures and the connection topologies affect the convergence more significantly.

With these observations, we conclude that the popular NAS architectures with wider and shallower cells indeed converge faster and more stably, which explains why these popular NAS cells are selected during



(a) CIFAR-10, 0.0025 (b) CIFAR-10, 0.25 (c) CIFAR-100, 0.0025 (d) CIFAR-100, 0.25

Figure 4.4: Test accuracy (%) curves of DARTS and its randomly connected variants on CIFAR-10 and CIFAR-100 during training under different learning rates (0.0025 and 0.25). We only evaluate C^{darts}, C_1^{darts} and C_3^{darts} for illustration. The caption of each sub-figure reports the dataset and the learning rate.



(a) DARTS (b) ENAS (c) AmoebaNet (d) NASNet

Figure 4.5: Test accuracy (%) curves of DARTS, ENAS, AmoebaNet, NASNet and their random variants of operations on CIFAR-10 during training. The parameter size is attached in Table 4.3 of Sec. 4.5.2.

the architecture search. The next question is then *why the wider and shallower cell leads to a faster and more stable convergence?*

4.3.2 Empirical Study of Factors Affecting Convergence

Since the wide and shallow cell is related to fast convergence, we further conduct the theoretical convergence analysis to investigate the cause of fast convergence. In this section, we first introduce the convergence analysis (i.e., Theorem 4.2) of non-convex optimization with the randomized stochastic gradient method (Ghadimi and Lan, 2013). Based on the analysis, we introduce the possible factors related to the common connection pattern that may affect the convergence. We then examine these factors empirically in the following subsections.

Theorem 4.2. *(Ghadimi and Lan, 2013) Let F be a L-smooth non-convex*

function, and let F^* be its minimal. Given repeated, independent accesses to stochastic gradients with variance bound σ^2 for $F(\boldsymbol{w})$, SGD with initial $\boldsymbol{w}_0$, total iterations $N > 0$ and learning rate $\eta_k < 1/L$ achieves the following convergence by randomly choosing $\boldsymbol{w}_k$ as the final output $\boldsymbol{w}_R$ with probability η_k/H where $H = \sum_{k=1}^{N} \eta_k$:

$$\mathbb{E}[\left\|\nabla F\left(\boldsymbol{w}_R\right)\right\|^2] \leq \frac{2(F\left(\boldsymbol{w}_0\right) - F^*)}{H} + \frac{L\sigma^2}{H} \sum_{k=1}^{N} \eta_k^2$$

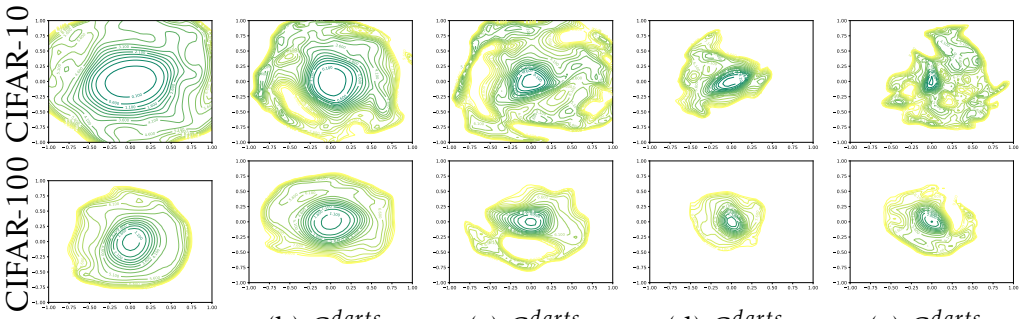
In this work, F and $\boldsymbol{w}$ denote the composition of model and objective (loss) function, and model parameters respectively. Based on the above theorem, Lipschitz smoothness L and gradient variance σ^2 significantly affect the convergence, including the rate and the stability of convergence. Particularly, given a specific number of iterations N, a smaller Lipschitz constant L or smaller gradient variance σ^2 would lead to a smaller convergence error and less damped oscillations, which indicates a faster and more stable convergence. Since the Lipschitz constant L and gradient variance σ^2 are highly related to the model we used (i.e., neural architecture in *deep learning*), different NAS architectures result in different L and σ^2. In the following subsections, we therefore conduct empirical analysis for the impacts of the cell with and depth on the Lipschitz smoothness and gradient variance.

4.3.2.1 Loss Landscape

The constant L of Lipschitz smoothness is closely correlated with the Hessian matrix of the objective function as shown by (Nesterov, 2004), which requires substantial computation and can only represent the global smoothness. The loss contour, which has been widely adopted to visualize the loss landscape of neural networks by (Goodfellow and Vinyals,

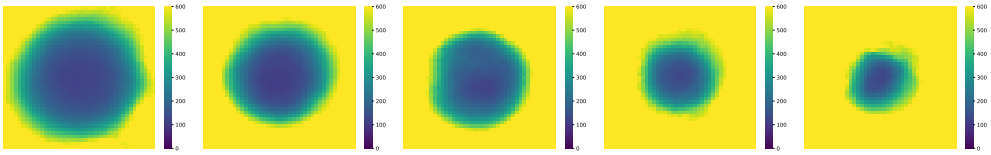
2015; Li et al., 2018), is instead computationally efficient and is able to report the local smoothness of the objective function. To explore the loss landscape of different architectures, we adopt the method in (Li et al., 2018) to plot the loss contour $s(\alpha, \beta) = \mathbb{E}_{i\sim P}[f_i(\boldsymbol{w}^* + \alpha \boldsymbol{w}_1 + \beta \boldsymbol{w}_2)]$. The notation $f_i(\cdot)$ denotes the loss evaluated at i_{th} instance in the dataset and P denotes the distribution of dataset. The notation $\boldsymbol{w}^*$, $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ denote the (local) optimal and two direction vectors randomly sampled from Gaussian distribution respectively. And α, β, which are the x and y axis of the plots, denote the step sizes to perturb $\boldsymbol{w}^*$. The loss contour plotted here is therefore a two-dimensional approximation of the truly high-dimensional loss contour. However, as shown in (Li et al., 2018), the approximation is valid and effective to characterize the property of the true loss contour.

To study the impact of the cell width and depth on Lipschitz smoothness, we compare the loss landscape between popular NAS architectures and their randomly connected variants trained in Section 4.3.1 on CIFAR-10 and CIFAR-100. Due to the space limitation, we only plot the loss landscape of DARTS (Liu et al., 2019) and its randomly connected variants in Figure 4.6. We observe that the connection topology has a significant influence on the smoothness of the loss landscape. With the widest and shallowest cell, C^{darts} has a fairly benign and smooth landscape along with the widest near-convex region around the optimal. With a deeper and narrower cell, C_1^{darts} and C_2^{darts} have a more agitated loss landscape compared with C^{darts}. Further, C_3^{darts}, with the smallest width and largest depth among these cells, has the most complicated loss landscape and the narrowest and steepest near-convex region around the optimum. The largest eigenvalue of the Hessian matrix, which indicates the maximum curvature of the objective function, is positively correlated with Lips-



(a) C^{darts} (b) C_1^{darts} (c) C_2^{darts} (d) C_3^{darts} (e) C_4^{darts}

Figure 4.6: Loss contours of DARTS and its variants with random connections on the test dataset of CIFAR-10. The lighter color of the contour lines indicates a larger loss. Notably, the loss of the blank area, around the corners of each plot, is extremely large. Besides, the area with denser contour lines indicates a steeper loss surface.

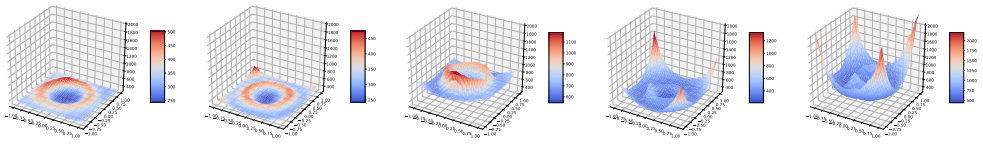


(a) C^{darts} (b) C_1^{darts} (c) C_2^{darts} (d) C_3^{darts} (e) C_4^{darts}

Figure 4.7: Heat maps of the gradient variance from DARTS and its randomly connected variants around the optimal on the test dataset of CIFAR-10. The lighter color indicates a larger gradient variance. Notably, the gradient variance of the yellow area, around the corners of each plot, is extremely large. Obviously, the region with relatively small gradient variance becomes smaller from left to right.

chitz constant as shown by (Nesterov, 2004). A smoother loss landscape therefore corresponds to a smaller Lipschitz constant L. C^{darts} is likely to achieve the smallest Lipschitz constant among these cells.

Consistent results can be found in Sec. 4.5.3 for the loss landscape of other popular NAS cells and their variants. Based on these results, we conclude that increasing the width and decreasing the depth of a cell widens the near-convex region around the optimal and smooths the loss landscape. The constant L of Lipschitz smoothness therefore becomes smaller locally and globally. Following Theorem 4.2, architectures with wider and shallower cells shall converge faster and more stably.



(a) C^{darts} (b) C_1^{darts} (c) C_2^{darts} (d) C_3^{darts} (e) C_4^{darts}

Figure 4.8: 3D surfaces of the gradient variance from DARTS and its randomly connected variants around the optimal on the test dataset of CIFAR-100. The height of the surface indicates the value of gradient variance. Notably, the height of the gradient variance surface is gradually increasing from left to right. Especially, C^{darts} has the smoothest and lowest surface of gradient variance among these architectures.

4.3.2.2 Gradient Variance

The gradient variance indicates the noise level of gradient by randomly selecting training instances in *stochastic gradient descent* (SGD) method. Large gradient variance indicates large noise in the gradient, which typically results in unstable updating of model parameters. Following (Ghadimi and Lan, 2013), gradient variance is defined as $\mathrm{Var}[\nabla f_i(\boldsymbol{w})]$. Similar to the visualization of loss landscape in Section 4.3.2.1, we visualize the gradient variance by $g(\alpha, \beta) = \mathrm{Var}[\nabla f_i(\boldsymbol{w}^* + \alpha \boldsymbol{w}_1 + \beta \boldsymbol{w}_2)]$. All other notations follow Section 4.3.2.1.

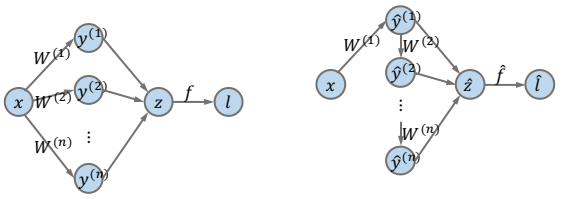
To study the impact of the width and depth of a cell on the gradient variance, we compare the gradient variance between popular NAS architectures and their randomly connected variants trained in Section 4.3.1 on CIFAR-10 and CIFAR-100. We visualize the gradient variance of DARTS (Liu et al., 2019) and its randomly connected variants in Figure 4.7 and Figure 4.8. For better visualization, we plot the figures using the standard deviation (i.e., $\sqrt{g(\alpha, \beta)}$) to avoid extremely large values in the visualization of DARTS. Obviously, as the cell width decreases and the cell depth increases (i.e., from C^{darts} to C_4^{darts}), the region with relatively small gradient variance becomes smaller as shown in Figure 4.7. Consistently, the gradient variance generally shows an increasing trend

from C^{darts} to C_4^{darts} in Figure 4.8. Consequently, the gradient becomes noisier in the neighborhood of the optimal, which typically makes the optimization harder and unstable.

Similar results from other popular NAS architectures and their random variants are provided in Sec. 4.5.4. Based on these results, we conclude that the increase in width and the decrease in depth of a cell result in a smaller gradient variance, which makes the optimization process less noisy and more efficient. The convergence of wide and shallow cells therefore shall be fast and stable following Theorem 4.2.

4.3.3 Theoretical Analysis of Factors Affecting Convergence

Our empirical study so far suggests that larger cell width and smaller cell depth smooth the loss landscape and decrease the gradient variance. Consequently, popular NAS architectures with wide and shallow cells converge fast. In this section, we investigate the impacts of the cell width and depth on Lipschitz smoothness and gradient variance from a theoretical perspective.



(a) case I: widest cell

(b) case II: narrowest cell

Figure 4.9: Two architectures to compare in the theoretical analysis: (a) architecture with widest cell; (b) architecture with narrowest cell. The notation l and $\widehat{l}$ denote the values of objective function f and $\widehat{f}$ evaluated at input $\boldsymbol{x}$ respectively.

4.3.3.1 Setup

We analyze the impact of the cell width and depth by comparing architectures with the widest cell and the narrowest cell as shown in Figure 4.9. To simplify the analysis, the cells we investigate contain only one input node $\boldsymbol{x}$ and one output node. The input node may be training instances or output node from any proceeding cell. All operations in the cell are linear operations without any non-linearity. Suppose there are n intermediate nodes in a cell, the i_{th} intermediate node and its associated weight matrix are denoted as $\boldsymbol{y}^{(i)}$ and $W^{(i)}(i = 1, \cdots, n)$ respectively. The output node $\boldsymbol{z}$ denotes the concatenation of all intermediate nodes. Both cells have the same arbitrary objective function f following the output node, which shall consist of the arbitrary number of activation functions and cells. For clarity, we refer to the objective function, intermediate nodes and output node of the architecture with the narrowest cell as $\widehat{f}$, $\widehat{\boldsymbol{y}}^{(i)}$ and $\widehat{\boldsymbol{z}}$ respectively. As shown in Figure 4.9, the intermediate node $\boldsymbol{y}^{(i)}$ and $\widehat{\boldsymbol{y}}^{(i)}$ can be computed by $\boldsymbol{y}^{(i)} = W^{(i)}\boldsymbol{x}$ and $\widehat{\boldsymbol{y}}^{(i)} = \prod_{k=1}^{i} W^{(k)}\boldsymbol{x}$ respectively. Particularly, we set $\prod_{k=1}^{i} W^{(k)} = W^{(i)}W^{(i-1)}\cdots W^{(1)}$. And all the related proofs of following theorems can be found in Appendix A.2.

4.3.3.2 Theoretical Results

Due to the complexity of the standard Lipschitz smoothness, we instead investigate the block-wise Lipschitz smoothness (Beck and Tetruashvili, 2013) of the two cases shown in Figure 4.9. In Theorem 4.3, we show that the block-wise Lipschitz constant of the narrowest cell is scaled by the largest eigenvalues of the model parameters (i.e., $W^{(i)}(i = 1, \cdots, n)$). Notably, the Lipschitz constant of the narrowest cell can be significantly larger than the one of the widest cell while most of the largest eigenvalues are larger than 1, which slows down the convergence substantially. The

empirical study in Section 4.3.2.1 has validated the results.

Theorem 4.3 (The impact of cell width and depth on block-wise Lipschitz smoothness). *Let $\lambda^{(i)}$ be the largest eigenvalue of $W^{(i)}$. Given the widest cell with objective function f and the narrowest cell with objective function $\widehat{f}$, by assuming the block-wise Lipschitz smoothness of the widest cell as $\left\| \frac{\partial f}{\partial W_1^{(i)}} - \frac{\partial f}{\partial W_2^{(i)}} \right\| \leq L^{(i)} \left\| W_1^{(i)} - W_2^{(i)} \right\|$ for any $W_1^{(i)}$ and $W_2^{(i)}$, the block-wise Lipschitz smoothness of the narrowest cell then can be represented as*

$$\left\| \frac{\partial \widehat{f}}{\partial W_1^{(i)}} - \frac{\partial \widehat{f}}{\partial W_2^{(i)}} \right\| \leq (\prod_{j=1}^{i-1} \lambda^{(j)}) L^{(i)} \left\| W_1^{(i)} - W_2^{(i)} \right\|$$

We then compare the gradient variance of the two cases shown in Figure 4.9. Interestingly, gradient variance suggests a similar but more significant difference between the two cases compared with their difference in Lipschitz smoothness. As shown in Theorem 4.4, the gradient variance of the narrowest cell is not only scaled by the square of the largest eigenvalue of the weight matrix but also is scaled by the number of intermediate nodes (i.e., n). Moreover, the upper bound of its gradient variance has numbers of additional terms, leading to a significantly larger gradient variance. The empirical study in Section 4.3.2.2 has confirmed the results.

Theorem 4.4 (The impact of cell width and depth on gradient variance). *Let $\lambda^{(i)}$ be the largest eigenvalue of $W^{(i)}$. Given the widest cell with objective function f and the narrowest cell with objective function $\widehat{f}$, by assuming the gradient variance of the widest cell as $\mathbb{E} \left\| \frac{\partial f}{\partial W^{(i)}} - \mathbb{E} \frac{\partial f}{\partial W^{(i)}} \right\|^2 \leq (\sigma^{(i)})^2$ for any $W^{(i)}$, the gradient variance of the narrowest cell is then bounded by*

$$\mathbb{E} \left\| \frac{\partial \widehat{f}}{\partial W^{(i)}} - \mathbb{E} \frac{\partial \widehat{f}}{\partial W^{(i)}} \right\|^2 \leq n \sum_{k=i}^{n} (\frac{\sigma^{(k)}}{\lambda^{(i)}} \prod_{j=1}^{k} \lambda^{(j)})^2$$

4.4 Generalization beyond the Common Connections

Our empirical and theoretical results so far have demonstrated that the common connection pattern helps to smooth the loss landscape and make the gradient more accurate. Popular NAS architectures with wider and shallower cells therefore converge faster, which explains why popular NAS architectures are selected by the NAS algorithms. Nonetheless, we have ignored the generalization performance obtained by popular NAS architectures and their random variants. We therefore wonder *whether popular NAS architectures with wide and shallow cells generalize better*.

In Figure 4.10, we visualize the test accuracy of popular NAS architectures and their randomly connected variants trained in Section 4.3.1. Notably, the popular NAS architectures can achieve competitive accuracy compared with most of the random variants. However, there are some random variants, which achieve higher accuracy than the popular architectures. Interestingly, there seems to be an optimal choice of depth and width for a cell to achieve higher test accuracy (i.e., C_7 for DARTS and C_4 for ENAS). Popular NAS architectures with wide and shallow cells therefore are not guaranteed to generalize better, although they typically converge faster than other random variants.

We also adapt the connections of popular NAS architectures to obtain their widest and shallowest variants. The adaptation is possible due to the fact that the cells (including normal and reduction cell) of popular NAS architectures are generally not widest and narrowest as shown in Figure 4.1. While there are various widest and shallowest cells following our definition of cell width and depth, we apply the connection pattern of SNAS cell shown in Figure 4.1(e) to obtain the widest and shallowest

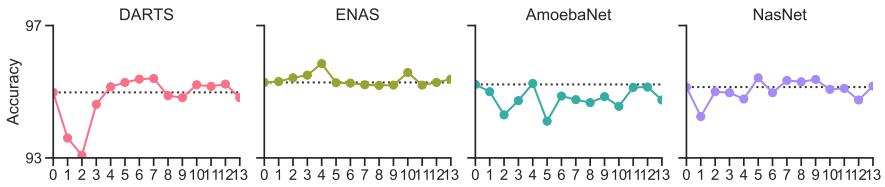


Figure 4.10: Comparison of the test accuracy at the convergence between popular NAS architectures and their randomly connected variants on CIFAR-10. Each popular NAS architecture (index 0 on the x-axis) is followed by 13 randomly connected variants (from index 1 to index 13 on the x-axis), corresponding to C_1 to C_{13} respectively. The width and depth of these random variants are shown in Table 4.2 in Sec. 4.5.2. The dashed lines report the accuracy of the popular NAS architectures.

Table 4.1: Comparison of the test error at the convergence between the original and the adapted NAS architectures on CIFAR-10/100 and Tiny-ImageNet-200. The entire networks are constructed and trained following the experimental settings reported in Appendix A.1.3, which may slightly deviate from the original ones. The test errors (or the parameter sizes) of original and adapted architectures are reported on the left and right hand-side of slash respectively.

Architecture	CIFAR-10		CIFAR-100		Tiny-ImageNet-200	
	Error(%)	Params(M)	Error(%)	Params(M)	Error(%)	Params(M)
NASNet (Zoph et al., 2018)	**2.65**/2.80	4.29/4.32	17.06/**16.86**	4.42/4.45	**31.88**/32.05	4.57/4.60
AmoebaNet (Real et al., 2019a)	**2.76**/2.91	3.60/3.60	17.55/**17.28**	3.71/3.71	**32.22**/33.16	3.83/3.83
ENAS (Pham et al., 2018)	**2.64**/2.76	4.32/4.32	16.67/**16.04**	4.45/4.45	**30.68**/31.36	4.60/4.60
DARTS (Liu et al., 2019)	**2.67**/2.73	3.83/3.90	16.41/**16.15**	3.95/4.03	**30.58**/31.33	4.08/4.16
SNAS (Xie et al., 2019b)	2.88/**2.69**	3.14/3.19	17.78/**17.20**	3.26/3.31	**32.40**/32.61	3.39/3.45

cells. The adapted topologies are shown in Figure 4.26 of Sec. 4.5.5.

Table 4.1 illustrates the comparison of the test accuracy between our adapted NAS architectures and the original ones. As shown in Table 4.1, the adapted architectures achieve smaller test error on CIFAR-100. Nevertheless, most of the adapted architectures, obtain larger test error than the original NAS architectures on both CIFAR-10 and Tiny-ImageNet-200[4]. The results again suggest that the widest and shallowest cells may not help architectures generalize better, while these architectures typically achieve compelling generalization performance.

The results above have revealed that the architectures with wide and shallow cells may not generalize better despite their fast convergence.

[4] https://tiny-imagenet.herokuapp.com/

To improve current NAS algorithms, we therefore need to rethink the evaluation of the performance of candidate architectures during architecture search since the current NAS algorithms are not based on the generalization performance at convergence as mentioned in Section 4.3.1. Nonetheless, architectures with the wide and shallow cells usually guarantee a stable and fast convergence along with competitive generalization performance, which should be good prior knowledge for designing architectures and NAS algorithms.

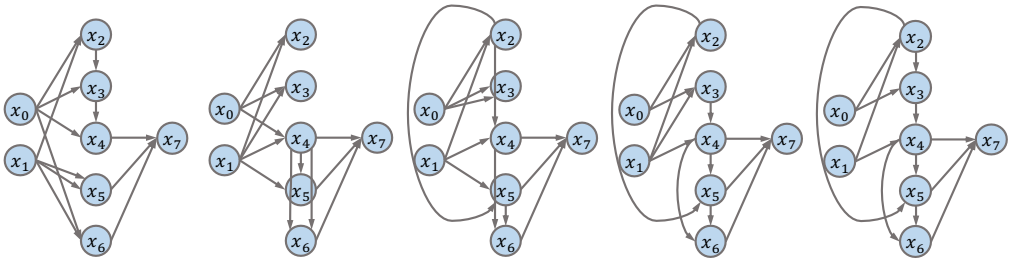
4.5 More Results

4.5.1 NAS architectures and Their Variants

We compare the width and depth of popular NAS architectures and their variants of random connections in Table 4.2. The random variants are sampled following the method in Appendix A.1.2. We further show the connection topologies of popular NAS and their partial random variants of connections in Figure 4.11 and Figure 4.12.

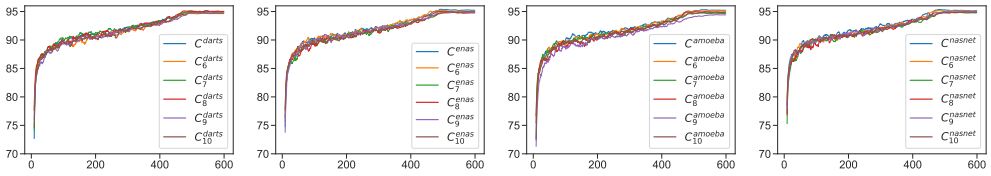
Table 4.2: Comparison of the width and depth of popular NAS cells and their randomly variants of connections. The name of the popular NAS cell is followed by its width and depth, which is separated by a comma. The width of a cell is conventionally computed by assuming that each intermediate node shares the same width c. Notably, the width and depth of random variants are in ascending and descending order respectively from C_1 to C_{13}. Moreover, the popular NAS architectures achieve the largest width and nearly the smallest depth among all the variants.

Base Cell	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
DARTS $(3.5c, 3)$	$2c, 4$	$2c, 4$	$2c, 4$	$2.5c, 4$	$2.5c, 4$	$2.5c, 3$	$2.5c, 3$	$2.5c, 3$	$3c, 3$	$3c, 3$	$3c, 3$	$3.5c, 3$	$3.5c, 3$
ENAS $(5c, 2)$	$1.5c, 6$	$1.5c, 5$	$2c, 6$	$2c, 6$	$2.5c, 5$	$2.5c, 5$	$3c, 4$	$3c, 3$	$3.5c, 5$	$3.5c, 4$	$3.5c, 4$	$3.5c, 3$	$3.5c, 3$
AmoebaNet $(4c, 4)$	$1.5c, 6$	$1.5c, 5$	$1.5c, 5$	$1.5c, 3$	$2c, 6$	$2c, 6$	$2c, 4$	$2.5c, 5$	$2.5c, 3$	$2.5c, 3$	$3c, 3$	$3.5c, 4$	$3.5c, 3$
NASNet $(5c, 2)$	$1.5c, 6$	$1.5c, 5$	$2c, 6$	$2c, 6$	$2.5c, 5$	$2.5c, 5$	$3c, 4$	$3c, 3$	$3.5c, 5$	$3.5c, 4$	$3.5c, 4$	$3.5c, 3$	$3.5c, 3$



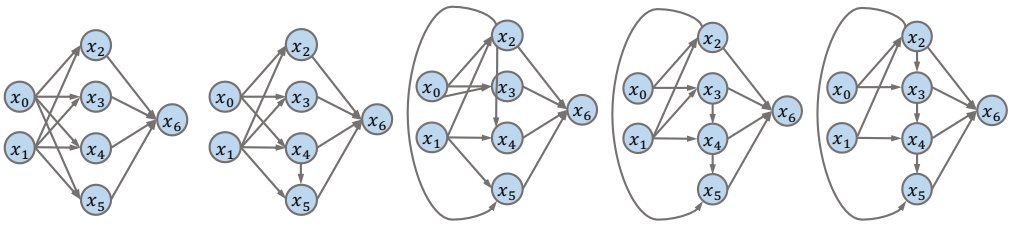
(a) 3.5c, 4 (b) 3.5c, 3 (c) 3c, 4 (d) 2.5c, 5 (e) 2c, 6

Figure 4.11: Connection topology of AmoebaNet cell (Real et al., 2019a) and its part of randomly connected variants. Each sub-figure reports the width and depth of a cell separated by a comma. The leftmost one is the original connection from AmoebaNet normal cell and others are the ones randomly sampled. The width of a cell is also computed by assuming that each intermediate node shares the same width c. Notably, the original AmoebaNet cell has the largest width and almost the smallest depth among these cells.



(a) DARTS (b) ENAS (c) AmoebaNet (d) NASNet

Figure 4.13: More test accuracy (%) curves of DARTS, ENAS, AmoebaNet, NASNet and their random variants of operations on CIFAR-10 during training.



(a) 4c, 2 (b) 3.5c, 3 (c) 3c, 3 (d) 2.5c, 4 (e) 2c, 5

Figure 4.12: Connection topology of SNAS cell under mild constraint (Xie et al., 2019b) and its part of randomly connected variants. The width and depth of a cell are reported in the title of each plot. The leftmost one is the original connection from SNAS normal cell and others are the ones randomly sampled. The width of a cell is conventionally computed by assuming that each intermediate node shares the same width c. Notably, the original SNAS cell has the largest width and the smallest depth among these cells.

Table 4.3: Comparison of the parameter size (MB) of popular NAS cells and their randomly variants of operations. C_0 denotes the original NAS cell and C_1 to C_{10} denote the random variants. Notably, there is a gap of $\sim 30\%$ between the parameter size of the smallest architecture and one of the largest architecture.

Base cell	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
DARTS	3.35	3.37	2.84	2.70	2.98	3.19	2.43	3.49	2.88	3.31	2.81
ENAS	3.86	3.45	3.19	2.98	2.70	3.67	3.03	3.85	3.26	3.81	3.29
AmoebaNet	3.15	2.86	2.62	2.41	2.10	3.10	2.46	3.28	2.69	3.42	2.75
NASNet	3.83	3.45	3.19	2.98	2.70	3.67	3.03	3.85	3.26	3.81	3.29

4.5.2 Convergence

In this section, we plot more test loss curves on CIFAR-10 (Krizhevsky et al., 2009) for original popular NAS architectures and their (12) randomly connected variants, as shown in Figure 4.14, Figure 4.15 and Figure 4.17. The depth and width of these 12 randomly connected variants can be found in Table 4.2. Notably, the width and depth of random variants (from C_1 to C_{12}) are in ascending and descending order respectively. Moreover, the popular NAS architectures achieve the largest width and nearly the smallest depth among all the variants. As shown in the following figures, the popular NAS cells, with larger width and smaller depth, typically achieve faster and more stable convergence than the random variants. Furthermore, with the increasing width and the decreasing depth, the convergence of random variants approaches to the original NAS architecture.

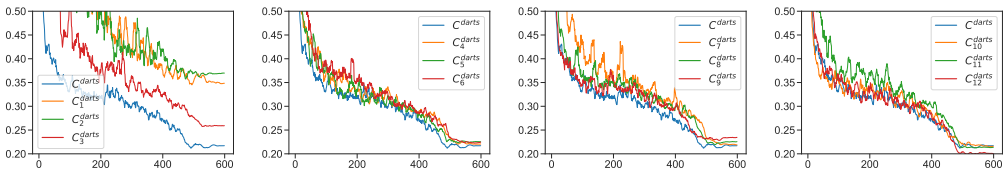


Figure 4.14: Test loss curves of DARTS and its variants on CIFAR-10 during training.

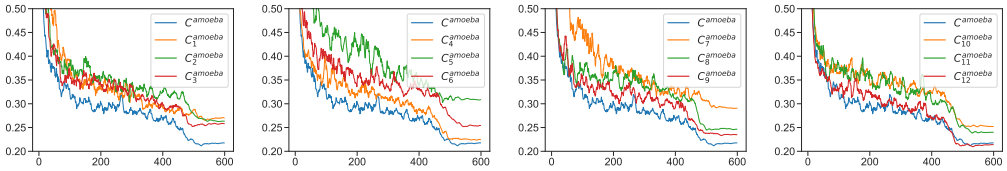


Figure 4.15: Test loss curves of AmoebaNet and its variants on CIFAR-10 during training.

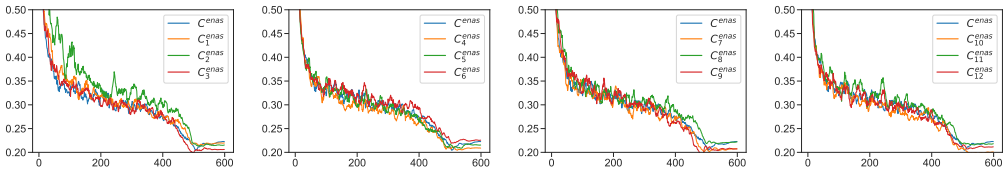


Figure 4.16: Test loss curves of ENAS and its variants on CIFAR-10 suring training.

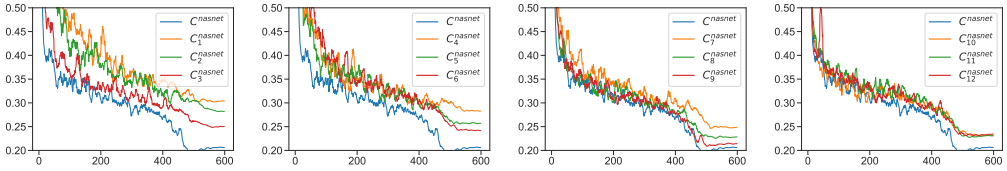
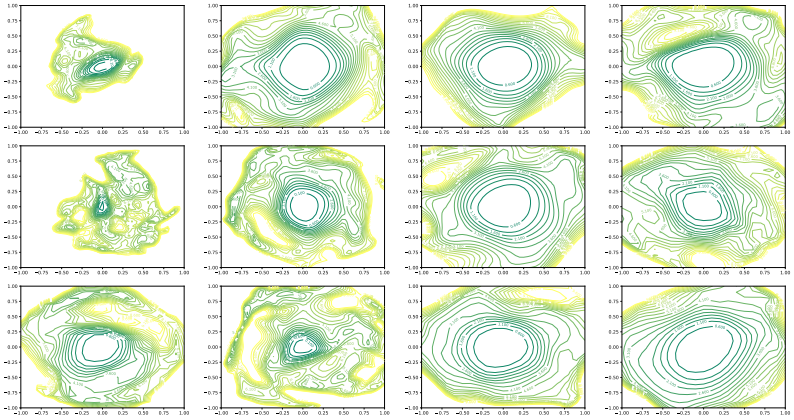


Figure 4.17: Test loss curves of NASNet and its variants on CIFAR-10 suring training.

4.5.3 Loss Landscape

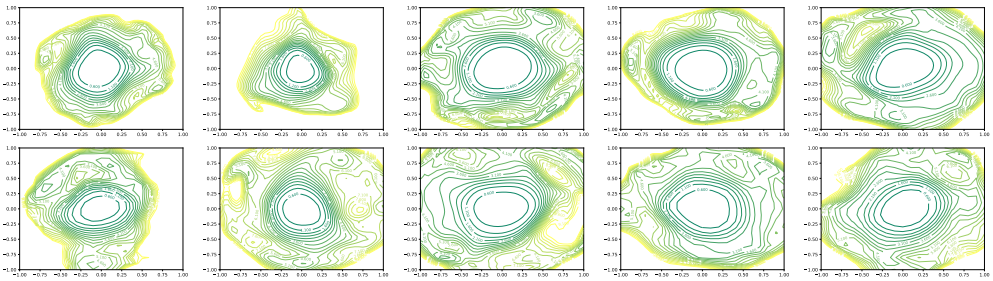
In this section, we visualize loss landscapes for popular NAS architectures and their randomly connected variants. The depth and width of a cell are highly correlated. For example, the depth and width cannot reach their maximum simultaneously. With the increasing width, the average depth of cells grouped by the same width is decreasing as shown in Table 4.2. We therefore only group the results (including the ones from original NAS architectures) with various width levels of a cell for a better comparison. Notably, the architectures with wider and shallower cells have a smoother and benigner loss landscape, as shown in Figure 4.18,

Figure 4.19, Figure 4.20 and Figure 4.21, which further supports the results in Section 4.3.2.1.



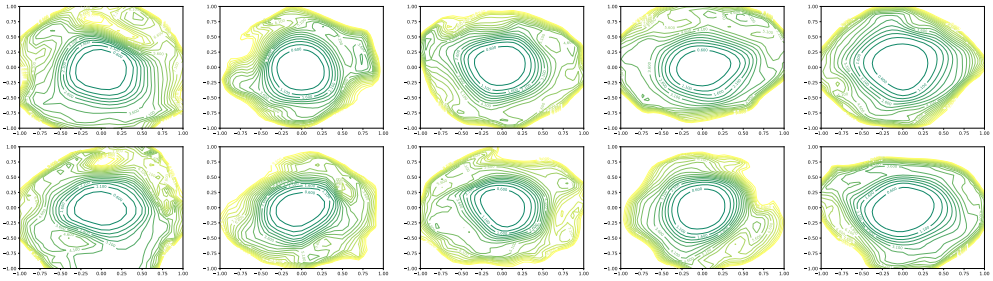
(a) $2c$ (b) $2.5c$ (c) $3c$ (d) $3.5c$

Figure 4.18: Loss contours of DARTS and its variants with random connections on the test dataset of CIFAR-10.



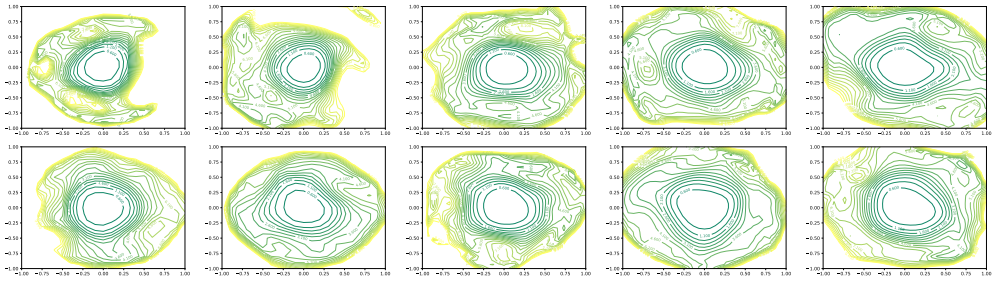
(a) $1.5c$ (b) $2c$ (c) $2.5c$ (d) $3c$ (e) $3.5c$

Figure 4.19: Loss contours of AmoebaNet and its randomly connected variants on the test dataset of CIFAR-10.



(a) $1.5c$ (b) $2c$ (c) $2.5c$ (d) $3c$ (e) $3.5c$

Figure 4.20: Loss contours of ENAS and its randomly connected variants on the test dataset of CIFAR-10.

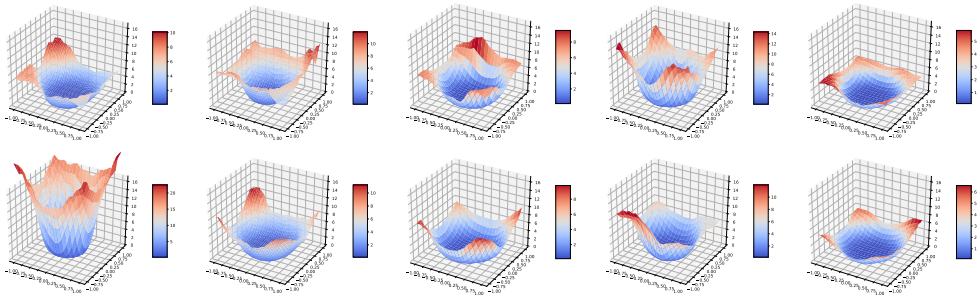


(a) 1.5c (b) 2c (c) 2.5c (d) 3c (e) 3.5c

Figure 4.21: Loss contours of NASNet and its randomly connected variants on the test dataset of CIFAR-10.

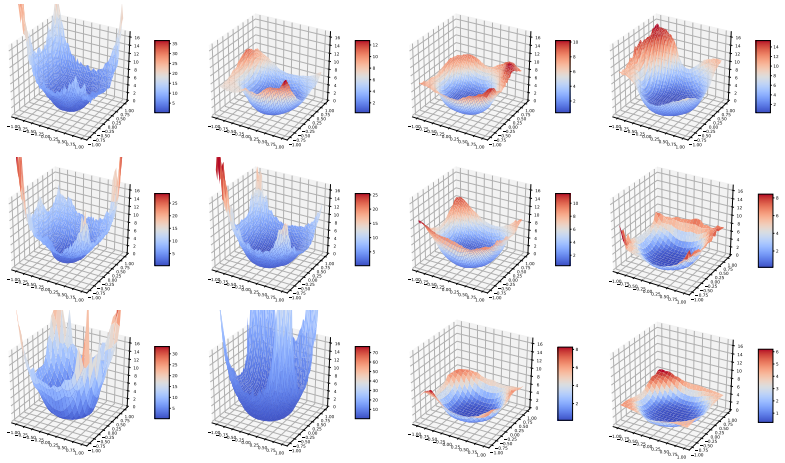
4.5.4 Gradient Variance

In this section, we visualize the gradient variance (i.e., $g(\alpha, \beta)$ as defined in Section 4.3.2.2) for the popular NAS architectures as well as their variants with random connection, such as AmoebaNet in Figure 4.22, DARTS in Figure 4.23, ENAS in Figure 4.24 and NASNet in Figure 4.24. The z-axis has been scaled by 10^{-5} for a better visualization. Similarly, we group the results based on the width of cells. Notably, architectures with wider and shallower cells achieve relatively smaller gradient variance, which further confirms the results in Section 4.3.2.2.



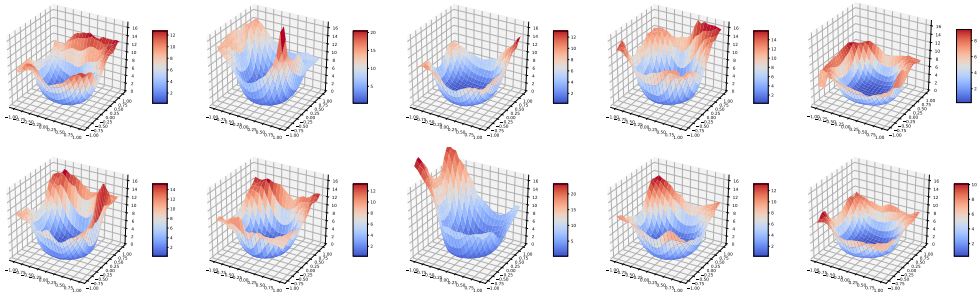
(a) 1.5c (b) 2c (c) 2.5c (d) 3c (e) 3.5c

Figure 4.22: 3D surfaces of the gradient variance from AmoebaNet and its randomly connected variants on the test dataset of CIFAR-10.



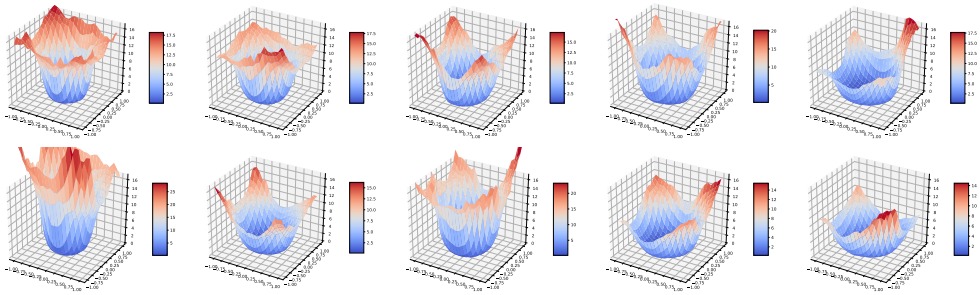
(a) 2c (b) 2.5c (c) 3c (d) 3.5c

Figure 4.23: 3D surfaces of the gradient variance from DARTS and its randomly connected variants on the test dataset of CIFAR-10.



(a) 1.5c (b) 2c (c) 2.5c (d) 3c (e) 3.5c

Figure 4.24: 3D surfaces of the gradient variance from ENAS and its randomly connected variants on the test dataset of CIFAR-10.

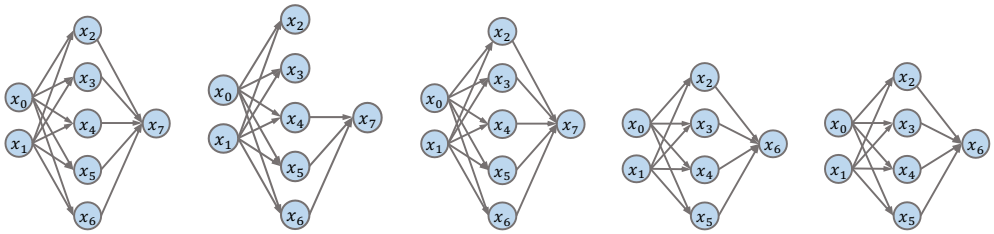


(a) 1.5c (b) 2c (c) 2.5c (d) 3c (e) 3.5c

Figure 4.25: 3D surfaces of the gradient variance from NASNet and its randomly connected variants on the test dataset of CIFAR-10.

4.5.5 Adapted Topologies

In this section, we visualize the adapted architectures (in Figure 4.26) we investigate on in Section 4.4. Notably, The adapted connection topologies are not only applied in the normal cell but also the reduction cell. The adapted architectures are compared with popular NAS architectures to examine the impacts of the common connection pattern on generalization.



(a) NASNet, 5c, 2 (b) AmoebaNet, 3c, 2 (c) ENAS, 5c, 2 (d) DARTS, 4c, 2 (e) SNAS, 4c, 2

Figure 4.26: Adapted topologies of cells from popular NAS architectures. The title of each sub-figure includes the name of the architecture, width and depth of the cell following our definition. Notably, these cells achieve the largest width and smallest depth in their original search space.

4.6 Conclusion and Discussion

Recent works have been focusing on the design and evaluation of NAS algorithms. We instead endeavour to examine the architectures selected by the various popular NAS algorithms. Our study is the first to explore the common structural patterns selected by existing algorithms, why these architectures are selected, and why these algorithms may be flawed. In particular, we reveal that popular NAS algorithms tend to favor architectures with wide and shallow cells, which typically converge fast and consequently are likely be selected during the search process. However, these architectures may not generalize better than other candidates of narrow and deep cells.

To further improve the performance of the selected NAS architectures, one promising direction for the current NAS research is to evaluate the generalization performance of candidate architectures more accurately and effectively. While popular NAS architectures appreciate fast and stable convergence along with competitive generalization performance, we believe that the wide and shallow cells are still useful prior knowledge for the design of the search space. We hope this work can attract more attention to the interpretation and understanding of existing popular NAS algorithms.

Chapter 5

Neural Ensemble Search via Bayesian Sampling

This chapter is based on the following work published at UAI 2022:
Shu, Y., Chen, Y., Dai, Z. & Low, B.K.H. (2022). Neural Ensemble Search via Bayesian Sampling. In *Proc. UAI-22*.

5.1 Introduction

Recent years have witnessed a surging interest in designing well-performing architectures for different tasks. These architectures are typically manually designed by human experts, which requires numerous trials and errors during this manual design process and therefore is prohibitively costly. Consequently, the increasing demand for developing well-performing architectures in different tasks makes this manual design infeasible. To avoid such human efforts, Zoph and Le (2017) have introduced *neural architecture search* (NAS) to help automate the design of architectures. Since then, a number of NAS algorithms (Pham et al., 2018; Liu et al., 2019; Chen et al., 2019) have been developed to improve the search ef-

ficiency (i.e., search cost) or the search effectiveness (i.e., generalization performance of their final selected architectures) in NAS.

However, conventional NAS algorithms aim to select only *one single architecture* from their search spaces and have thus overlooked the capability of other candidate architectures from the same search spaces in helping improve the performance achieved by their final selected single architecture. That is, *neural network ensembles* are widely known to be capable of achieving an improved performance compared with a single neural network in practice (Cortes et al., 2017; Gal and Ghahramani, 2016; Lakshminarayanan et al., 2017). This naturally begs the question: *How to select best-performing neural network ensembles with diverse architectures from a NAS search space in order to improve the performances achieved by existing NAS algorithms?* To the best of our knowledge, only limited efforts (e.g., (Zaidi et al., 2020)) have been devoted to this problem in the NAS literature. Unfortunately, the *neural ensemble search* (NES) algorithm based on random search or evolutionary algorithm in (Zaidi et al., 2020) requires excessive search costs to select their final neural network ensembles, which will not be affordable in resource-constrained scenarios.

To this end, this work introduces a novel algorithm, namely *neural ensemble search via Bayesian sampling* (NESBS), to effectively and efficiently select the well-performing neural network ensemble with diverse architectures from a search space. We firstly represent the search space as a supernet following conventional one-shot NAS algorithms and then use the model parameters inherited from this supernet after its model training to estimate the single-model performances and also the ensemble performance of independently trained architectures (Sec. 5.2.1). Next, since both single-model performances and diverse model predictions

affect the final ensemble performance according to (Zhou, 2012), we propose to use a variational posterior distribution of architectures based on a trained supernet to characterize these two factors, i.e., single-model performances and diverse model predictions (Sec. 5.2.2). We then introduce two novel Bayesian sampling algorithms based on the posterior distribution of architectures, i.e., *Monte Carlo sampling* (MC Sampling) and *Stein Variational Gradient Descent with regularized diversity* (SVGD-RD), to effectively and efficiently select ensembles with both competitive single-model performances and compelling diverse model predictions (Sec. 5.2.3), which is also guaranteed to be able to achieve impressive ensemble performances (Zhou, 2012). Lastly, we use extensive experiments to show that our NESBS algorithm is indeed able to select well-performing neural network ensembles effectively and efficiently in practice (Sec. 5.4).

5.2 Neural Ensemble Search via Bayesian Sampling

Let $f_{\mathcal{A}}(\boldsymbol{x}, \boldsymbol{\theta}_{\mathcal{A}})$ denote the output of an architecture $\mathcal{A}$ with input data $\boldsymbol{x}$ and model parameter $\boldsymbol{\theta}_{\mathcal{A}}$. Let S and $\boldsymbol{\Theta}_S$ denote a set of architectures and their corresponding model parameters, respectively. Given the ensemble scheme $\mathcal{F}_S(\boldsymbol{x}, \boldsymbol{\Theta}_S) \triangleq 1/n \sum_{\mathcal{A} \in S} f_{\mathcal{A}}(\boldsymbol{x}, \boldsymbol{\theta}_{\mathcal{A}})$ with an ensemble size of $|S| = n$,[1] let $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ denote the training and validation loss respectively, *Neural Ensemble Search* (NES) (Zaidi et al., 2020) can then be framed as a

[1] We apply such an ensemble scheme for simplicity following the work of (Zaidi et al., 2020).

bi-level combinatorial optimization problem:

$$\begin{aligned} &\min_{S} \mathcal{L}_{\text{val}}(\mathcal{F}_S(\boldsymbol{x}, \boldsymbol{\Theta}_S^*)) \\ &\text{s.t. } \forall\ \boldsymbol{\theta}_{\mathcal{A}}^* \in \boldsymbol{\Theta}_S^*,\ \boldsymbol{\theta}_{\mathcal{A}}^* = \arg\min_{\boldsymbol{\theta}_{\mathcal{A}}} \mathcal{L}_{\text{train}}(f_{\mathcal{A}}(\boldsymbol{x}, \boldsymbol{\theta}_{\mathcal{A}})) \,. \end{aligned} \tag{5.1}$$

Note that (5.1) is challenging to solve due to the following reasons: (I) The enormous number of candidate architectures in the NAS search space, e.g., $\sim 10^{25}$ in the DARTS search space (Liu et al., 2019), makes the independent model training of every candidate architecture (i.e., the lower-level optimization in (5.1)) unaffordable. (II) The ensemble search space is exponentially increasing in the ensemble size n, e.g., there are $\sim m^n$ different ensembles given m diverse architectures. The combinatorial optimization problem (i.e., the upper-level optimization in (5.1)) is thus intractable to solve within this huge ensemble search space. Recently, Zaidi et al. (2020) have attempted to avoid these two problems by selecting a small fraction of candidate architectures from the original search space for their final ensemble search. Nonetheless, they fail to explore the whole search space and therefore may only achieve limited ensemble performances. Furthermore, their search cost is unaffordable due to the independent model training of every selected architecture.

To this end, we novelly present the *neural ensemble search via Bayesian sampling* (NESBS) algorithm to solve (5.1) effectively and efficiently. We firstly employ the model parameters inherited from a supernet (i.e., a representation of the NAS search space) after its model training to estimate the single-model performances and also the ensemble performance of independently trained architectures (Sec. 5.2.1). This only requires the model training of the supernet and thus allows us to overcome the aforementioned challenge I. We then derive a posterior distribution of

architectures to characterize both the single-model performances and the diverse model predictions of candidate architectures in the search space (Sec. 5.2.2). Finally, based on this posterior distribution and also the aforementioned ensemble performance estimation, we introduce *Monte Carlo Sampling* (MC Sampling) and *Stein Variational Gradient Descent with regularized diversity* (SVGD-RD) to explore the ensembles in the whole search space effectively and efficiently (Sec. 5.2.3), which thus allows us to overcome the aforementioned challenge II. An overview of our NESBS is in Algorithm 5.1.

5.2.1 Model Training of Supernet

Similar to one-shot NAS algorithms (Liu et al., 2019; Pham et al., 2018), we represent NAS search space as a supernet. This then allows us to use the model parameters inherited from this trained supernet to estimate not only the single-model performances but also the ensemble performance of independently trained candidate architectures in the search space. However, in order to realize an accurate and fair estimation of these performances, we need to further ensure that every candidate architecture in the search space is trained for a comparable number of steps, namely, the training fairness among candidate architectures (Chu et al., 2019). To achieve this, in every training step of this supernet, we uniformly randomly sample one single candidate architecture from this supernet for model training (see Fig. 5.1). The training fairness of such a training scheme can then be theoretically guaranteed, as demonstrated in Appendix B.1. Moreover, we provide empirical results in Sec. 5.5.1 to validate the effectiveness of such performance estimations.

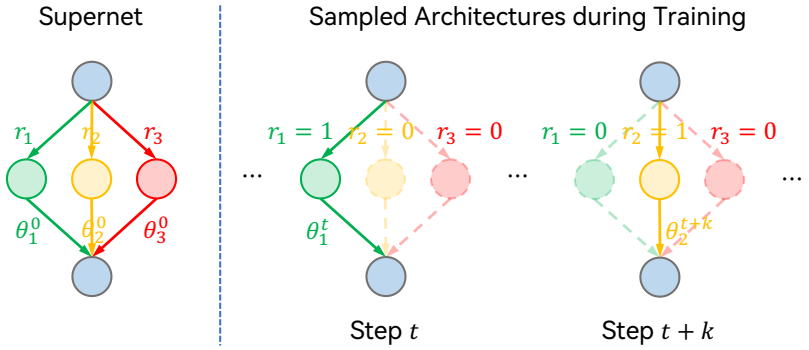


Figure 5.1: An illustration of the model training of supernet. The supernet here consists of three candidate architectures with r_i indicating the selection of one architecture and θ_i^t denoting its model parameters at step t. In every training step, only one architecture is uniformly sampled to update its parameters and all other architectures will be ignored.

5.2.2 Distribution of Architectures

It has been demonstrated that both competitive single-model performances and diverse model predictions are required to achieve compelling ensemble performances (Zhou, 2012). That is, NES algorithms should be capable of selecting architectures with both competitive single-model performances and diverse model predictions to achieve competitive ensemble performances. To realize this, we introduce a posterior distribution of architectures to firstly characterize these two factors. Let $\mathcal{D}$ denote the validation dataset, and $p(\mathcal{A})$ and $p(\mathcal{A}|\mathcal{D})$ denote, respectively, the prior and posterior distributions of a candidate architecture after its model training where $p(\mathcal{A})$ follows from a categorical uniform distribution, as required in Sec. 5.2.1. According to the Bayes' theorem, since $p(\mathcal{A})$ is uniform and $p(\mathcal{D})$ is constant,

$$p(\mathcal{A}|\mathcal{D}) = p(\mathcal{D}|\mathcal{A})p(\mathcal{A})/p(\mathcal{D}) \propto p(\mathcal{D}|\mathcal{A}) \tag{5.2}$$

where $p(\mathcal{D}|\mathcal{A})$ (i.e., likelihood) is widely used to represent the single-model performance (i.e., loss) in practice. So, (5.2) implies that the

posterior distribution $p(\mathcal{A}|\mathcal{D})$ can also characterize the single-model performances of architectures.

Meanwhile, given a γ-Lipschitz continuous loss function $\mathcal{L}(\boldsymbol{f})$, the diversity of model predictions (i.e., $\|\boldsymbol{f}_{\mathcal{A}_1} - \boldsymbol{f}_{\mathcal{A}_2}\|_2$) can then be lower bounded based on the Lipschitz continuity of $\mathcal{L}(\cdot)$:

$$\|\boldsymbol{f}_{\mathcal{A}_1} - \boldsymbol{f}_{\mathcal{A}_2}\|_2 \geq \gamma^{-1} |\mathcal{L}(\boldsymbol{f}_{\mathcal{A}_1}) - \mathcal{L}(\boldsymbol{f}_{\mathcal{A}_2})| \,. \tag{5.3}$$

Therefore, (5.3) suggests that in addition to being able to characterize the single-model performances of architectures (i.e., $\mathcal{L}(\boldsymbol{f})$), the posterior distribution $p(\mathcal{A}|\mathcal{D})$ can estimate the diversity of model predictions for different architectures (e.g., $\mathcal{A}_1$ and $\mathcal{A}_2$) using $|p(\mathcal{A}_1|\mathcal{D}) - p(\mathcal{A}_2|\mathcal{D})|$.

However, it is intractable to obtain exact posterior distribution $p(\mathcal{A}|\mathcal{D})$ in the NAS search space. So, we approximate it with a variational distribution $p_{\boldsymbol{\alpha}}(\mathcal{A})$ (parameterized by a low-dimensional $\boldsymbol{\alpha}$) that can be optimized via variational inference, i.e., by minimizing the KL divergence between $p_{\boldsymbol{\alpha}}(\mathcal{A})$ and $p(\mathcal{A}|\mathcal{D})$. Equivalently, we only need to maximize a lower bound of the log-marginal likelihood (i.e., the *evidence lower bound* (ELBO) (Kingma and Welling, 2014)) to get an optimal variational distribution $p_{\boldsymbol{\alpha}^*}(\mathcal{A})$:

$$\max_{\boldsymbol{\alpha}} \mathbb{E}_{\mathcal{A}\sim p_{\boldsymbol{\alpha}}(\mathcal{A})} \Big[\log p(\mathcal{D}|\mathcal{A})\Big] - \mathrm{KL}[p_{\boldsymbol{\alpha}}(\mathcal{A}) \| p(\mathcal{A})] \,. \tag{5.4}$$

Similar to (Kingma and Welling, 2014), a gradient-based optimization algorithm with the reparameterization trick is employed to solve (5.4) efficiently (see Sec. 5.3.3). While Xie et al. (2019b) have adopted a similar form to (5.4) (without the KL term) *during* the model training of the supernet (namely, the *best-response* posterior distribution), our *post-training* posterior distribution is able to not only provide a more accurate

characterization of the single-model performances but also contribute to an improved ensemble search performance, as demonstrated in Sec. 5.5.2.

5.2.3 Bayesian Sampling

To solve (5.1) effectively and efficiently, we finally introduce two novel Bayesian sampling algorithms based on the posterior distribution of architectures in Sec. 5.2.2, i.e., *Monte Carlo sampling* (MC Sampling) and *Stein Variational Gradient Descent with regularized diversity* (SVGD-RD), to sample ensembles with both competitive single-model performances and compelling diversity of model predictions, as required by well-performing ensembles (Zhou, 2012).

5.2.3.1 Monte Carlo Sampling (MC Sampling)

Given the posterior distribution of neural architectures obtained in Sec. 5.2.2, we firstly propose to employ *Monte Carlo sampling* (MC Sampling) to sample a set of neural architectures from this posterior distribution (Algorithm 5.2). Specifically, MC Sampling guarantees that neural architectures with better single-model performances will be sampled (i.e., exploited) with higher probabilities. Meanwhile, architectures with diverse model predictions can also be sampled (i.e., explored) due to the inherent randomness in the sampling process. Compared with conventional NAS algorithms that only select a single best-performing architecture in the search space (Dong and Yang, 2019b; Xie et al., 2019b), our MC sampling algorithm further extends these algorithms by exploring the capability of diverse neural architectures while preserving its exploitation of neural architectures with compelling single-model performances.

Algorithm 5.1 NES via Bayesian Sampling (NESBS)

Input: Iterations T, ensemble size n, a supernet
Train the supernet to get its tuned parameters $\boldsymbol{\theta}^*$
Obtain the posterior distribution $p_{\boldsymbol{\alpha}^*}(\mathcal{A})$ with (5.4)
for iteration $t = 1, \ldots, T$ **do**
    Sample S_t of size n via Algorithm 5.2 or 5.3
    Evaluate estimated $\mathcal{L}_{\text{val}}(\mathcal{F}_{S_t}(\boldsymbol{x}, \boldsymbol{\Theta}^*_{S_t}))$ given $\boldsymbol{\theta}^*$
Select optimum $S^* = \arg\min_{S_t} \mathcal{L}_{\text{val}}(\mathcal{F}_{S_t}(\boldsymbol{x}, \boldsymbol{\Theta}^*_{S_t}))$

Algorithm 5.2 MC Sampling

Input: Ensemble size n, set $S = \emptyset$, posterior $p_{\boldsymbol{\alpha}^*}(\mathcal{A})$
for iteration $i = 1, \ldots, n$ **do**
    Sample $\mathcal{A}_i \sim p_{\boldsymbol{\alpha}^*}(\mathcal{A})$
    $S \leftarrow S \cup \{\mathcal{A}_i\}$
Output: S

Algorithm 5.3 SVGD-RD

Input: Diversity coefficient δ, ensemble size n, iterations L, initial particles $\{\boldsymbol{x}_i^{(0)}\}_{i=1}^n$, posterior $p_{\boldsymbol{\alpha}^*}(\mathcal{A})$, kernel $k(\boldsymbol{x}, \boldsymbol{x}')$, step size $\{\epsilon_l\}_{l=1}^L$
for iteration $l = 0, \ldots, L-1$ **do**
    $\widehat{\boldsymbol{\phi}}_l^*(\boldsymbol{x}) = \frac{1}{n}\sum_{j=1}^{n} \nabla_{\boldsymbol{x}_j^{(l)}} k(\boldsymbol{x}_j^{(l)}, \boldsymbol{x}) - \delta \nabla_{\boldsymbol{x}} k(\boldsymbol{x}_j^{(l)}, \boldsymbol{x}) + k(\boldsymbol{x}_j^{(l)}, \boldsymbol{x}) \nabla_{\boldsymbol{x}_j^{(l)}} \log p_{\boldsymbol{\alpha}^*}$
    $\boldsymbol{x}_i^{(l+1)} \leftarrow \boldsymbol{x}_i^{(l)} + \epsilon_l \, \widehat{\boldsymbol{\phi}}_l^*(\boldsymbol{x}_i^{(l)})$
Output: $S = \{\mathcal{A}_i\}_{i=1}^n$ derived based on $\{\boldsymbol{x}_i^{(L)}\}_{i=1}^n$

5.2.3.2 SVGD with Regularized Diversity (SVGD-RD)

However, the diversity of sampled architectures using the MC Sampling algorithm above cannot be controlled and hence may lead to poor ensemble search results. So, in order to achieve a controllable diversity, we resort to *Stein Variational Gradient Descent* (SVGD). Theoretically, SVGD is capable of sampling particles with both large probability density and good diversity where the diversity is explicitly encouraged (i.e., by the second term in (2.8)). Nonetheless, in practice, the particles sampled by SVGD may still fail to represent the target distribution well owing to the lack of diversity among those sampled particles, as observed in (Zhuo

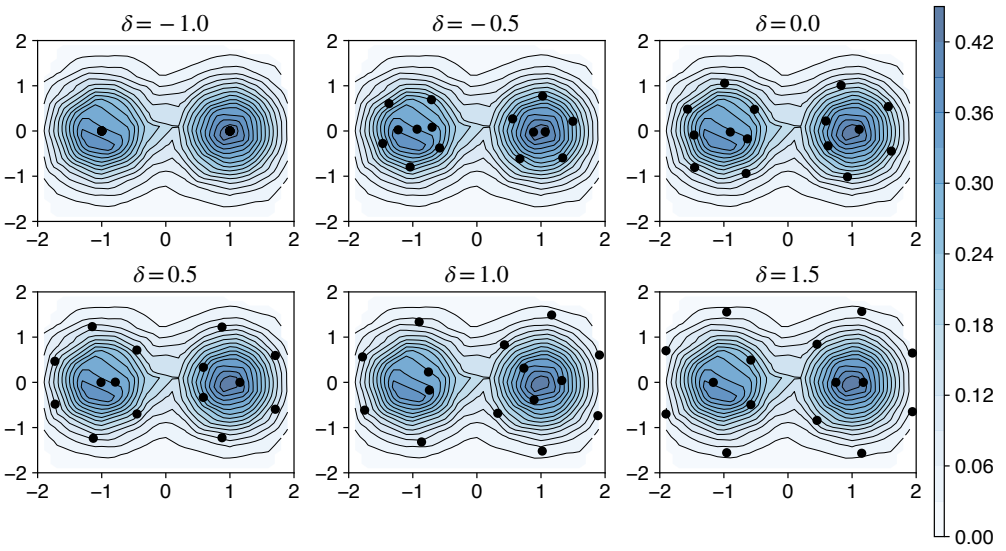


Figure 5.2: The impact of δ in SVGD-RD algorithm. We use contours and dots to denote the density of target distribution and sampled particles, respectively. The target distribution is chosen to be $p(\boldsymbol{x})=1/Z\left[\mathcal{N}(\boldsymbol{x}|\boldsymbol{u}_1,\Sigma_1)+\mathcal{N}(\boldsymbol{x}|\boldsymbol{u}_2,\Sigma_2)\right]$, where $\boldsymbol{u}_1=(-1,0)$, $\boldsymbol{u}_2=(0,1)$, $\Sigma_1=\Sigma_2=\text{diag}((0.25,0.5))$ and Z denotes the normalization constant. Those sampled particles are obtained from our Algorithm 5.3 using L=1000, n=15, ϵ_l=0.1 and a *radial basis function* (RBF) kernel. Notably, our sampled particles tend to become more diverse as δ is increased, indicating a controllable (via δ) diversity of the final sampled particles in our SVGD-RD algorithm. Meanwhile, SVGD-RD can consistently sample particles with high probability densities with varying δ.

et al., 2018). Besides, the diversity of sampled particles in standard SVGD still cannot be controlled by human experts.

We hence develop an *SVGD with regularized diversity* (SVGD-RD) sampling algorithm that can achieve a controllable diversity among those sampled particles. We follow the notations from Sec. 2.3. In particular, when optimizing the distribution q^* (represented by the n particles $\{\boldsymbol{x}_i^*\}_{i=1}^n$), we modify the objective in (2.4) by adding a term representing the (controllable) diversity among the particles measured by the kernel function $k(\boldsymbol{x},\boldsymbol{x}')$:

$$q^* = \arg\min_{q\in\mathcal{Q}} \text{KL}(q\|p) + n\delta\, \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim q}\left[k(\boldsymbol{x},\boldsymbol{x}')\right] \tag{5.5}$$

where δ is the parameter explicitly controlling the diversity, and p in (5.5) denotes the posterior distribution $p_{\alpha^*}(\mathcal{A})$ derived in Sec. 5.2.2 which we intend to sample from. Following the work of SVGD, q^* in (5.5) is represented by $\{\boldsymbol{x}_i^*\}_{i=1}^n$ denoting our final selected neural network ensemble that can achieve both competitive single-model performances (i.e., large probability density) and also diverse model predictions. Proposition 5.1 below provides one possible update rule for the particles $\{\boldsymbol{x}_i\}_{i=1}^n$ to optimize (5.5) (see its proof in Appendix B.1). Finally, Algorithm 5.3 summarizes the details of our SVGD-RD algorithm and Sec. 5.3.4 provides its optimization details in practice. After obtaining those optimal particles $\{\boldsymbol{x}_i^*\}_{i=1}^n$ in our SVGD-RD algorithm, we then apply these particles to derive the architectures in our final selected ensembles (see details in Sec. 5.3.4).

Proposition 5.1. *Given the proximal operator* $\text{prox}_h(\boldsymbol{y}) = \arg\min_{\boldsymbol{z}} h(\boldsymbol{z}) + 1/2\|\boldsymbol{z}-\boldsymbol{y}\|_2^2$, *by applying proximal gradient method (Parikh and Boyd, 2014) and proper approximation,* (5.5) *can be optimized via the following updates of the particles* $\{\boldsymbol{x}_i\}_{i=1}^n$*:*

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \frac{1}{n}\sum_{j=1}^{n} k(\boldsymbol{x}_j, \boldsymbol{x}_i)\nabla_{\boldsymbol{x}_j} \log p(\boldsymbol{x}_j) + \nabla_{\boldsymbol{x}_j} k(\boldsymbol{x}_j, \boldsymbol{x}_i) - \delta \nabla_{\boldsymbol{x}_i} k(\boldsymbol{x}_j, \boldsymbol{x}_i)\,.$$

Compared with MC Sampling, our SVGD-RD algorithm provides a controllable trade-off between the single-model performances and the diverse model predictions. On the one hand, the minimization of the KL divergence term in (5.5) encourages the selection of architectures with competitive single-model performances by favoring particles with high probability densities, as shown by Proposition 5.2 below (its proof is in

Appendix B.1).[2] On the other hand, the maximization of the scaled distance $-n\delta\ \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim q}\left[k(\boldsymbol{x},\boldsymbol{x}')\right]$ among the sampled particles leads to a controllable diversity (via δ) among these sampled particles and also a controllable diversity of the probability densities among these particles (see Fig. 5.2), which also implies a controllable diversity of the model predictions, as suggested in Sec. 5.2.2.

Proposition 5.2. *Let p be a target density and $k(\boldsymbol{x},\boldsymbol{x}') = c$ for every $\boldsymbol{x} = \boldsymbol{x}'$ where c is a constant. For any $\delta \in \mathbb{R}$, our SVGD-RD algorithm is equivalent to the maximization of the density p w.r.t. $\boldsymbol{x}$ in the case of $n = 1$.*

5.3 Experimental Settings

5.3.1 The Search Space

We use the DARTS (Liu et al., 2019) search space in our work: Each candidate architecture in the search space consists of a stack of L cells, which can be represented as a directed acyclic graph (DAG) of N nodes denoted by $\{z_0, z_1, \ldots, z_{N-1}\}$. Among these N nodes in a cell, z_0 and z_1 denote the input nodes produced by two preceding cells, and z_N denotes the output of a cell, which is the concatenation of all intermediate nodes, i.e., from z_2 to z_{N-1}. As in the work of Liu et al. (2019), to select the best-performing architectures, we need to select their corresponding cells, including the normal and reduction cell. We refer to DARTS (Liu et al., 2019) for more details. In practice, this search space is represented as a supernet stacked by 8 cells (6 normal cells and 2 reduction cells) with initial channels of 16.

[2] Although Proposition 5.2 is only applicable in the case of $n = 1$, our SVGD-RD is still capable of sampling particles with high probability densities when $n > 1$, as validated in Fig. 5.2.

5.3.2 The Model Training of Supernet

Following (Xu et al., 2020), we apply a partial channel connection with $K = 2$ in the model training of the supernet, which allows us to accelerate and reduce the GPU memory consumption during this model training. We split the standard training dataset of CIFAR-10 into two piles in our ensemble search: 70% randomly sampled data is used in the model training of the supernet, and the rest is used to obtain the posterior distribution of neural architectures in Sec. 5.2.2 and also the final selected ensembles in Sec. 5.2.3. To achieve a fair and sufficient model training of each candidate architecture, we adopt *stochastic gradient descent* (SGD) with epoch 50, learning rate cosine scheduled from 0.1 to 0, momentum 0.9, weight decay 3×10^{-4} and batch size 128 in the model training of the supernet, where one candidate architecture is uniformly randomly sampled from this supernet in each training step.

5.3.3 Posterior Distribution

Variational posterior distribution. Following (Xie et al., 2019b), the variational posterior distribution of architectures is represented as $p_{\boldsymbol{\alpha}}(\mathcal{A})$ parameterized by $\boldsymbol{\alpha}$. Specifically, within the search space demonstrated in our Sec. 5.3.1, each intermediate nodes z_i is the output of one selected operation $o \sim p_{\boldsymbol{\alpha}_i}(o)$ using the inputs from its proceeding nodes or cells, where $\mathcal{O}$ is a predefined operation set for our search. Specifically, given $\boldsymbol{\alpha}_i = (\alpha_i^{o_1} \cdots \alpha_i^{o_{|\mathcal{O}|}})$, $p_{\boldsymbol{\alpha}_i}(o)$ can be represented as

$$p_{\boldsymbol{\alpha}_i}(o) = \frac{\exp(\alpha_i^o/\tau)}{\sum_{o \in \mathcal{O}} \exp(\alpha_i^o/\tau)}, \tag{5.6}$$

where τ denotes the softmax temperature, which is usually set to be 1 in practice. Based on this defined probability for each intermediate node z_i,

our variational posterior distribution can be framed as

$$p_{\boldsymbol{\alpha}}(A) = \prod_{i=2}^{N-2} p_{\boldsymbol{\alpha}_i}(o) \,. \tag{5.7}$$

More precisely, this representation is applied for single-path architecture with identical cells. We use it to ease our representation. For double-path architectures consisting of two different cells (i.e., normal and reduction cell), e.g., the candidate architecture in the DARTS search space, a similar representation can be obtained.

Optimization details. To optimize (5.4), we firstly relax our variational posterior distribution to be differentiable using the Straight-Through (ST) Gumbel-Softmax (Maddison et al., 2017; Jang et al., 2017) with the reparameterization trick. More precisely, we propose a variant of ST Gumbel-Softmax outputting the double-path architectures in the DARTS search space. Then, we use stochastic gradient-based algorithms to optimize (5.4) efficiently. In each optimization step, we sample one neural architecture from the distribution $p_{\boldsymbol{\alpha}}(\mathcal{A})$ to estimate $\mathbb{E}_{\mathcal{A}\sim p_{\boldsymbol{\alpha}}(\mathcal{A})}\left[\log p(\mathcal{D}|\mathcal{A})\right]$ (i.e., the commonly used Cross-Entropy loss). In practice, we use Adam (Kingma and Ba, 2015) with learning rate 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay 3×10^{-4} to update our variational posterior distribution $p_{\boldsymbol{\alpha}}(\mathcal{A})$ for 20 epochs.

5.3.4 SVGD of Regularized Diversity

Continuous relaxation of variational posterior distribution. Notably, SVGD (Liu and Wang, 2016) and also our SVGD-RD is applied for continuous distribution. Unfortunately, the variational posterior distribution $p_{\boldsymbol{\alpha}}(\mathcal{A})$ is discrete due to a discrete search space. To apply SVGD-RD, we

firstly relax this discrete posterior into its continuous counterpart using a mixture of Gaussian distribution. Specifically, we represent each operation $o \in \mathcal{O}$ in (5.6) into a one-hot vector $\boldsymbol{h}_o$. By introducing the random variable $\boldsymbol{o}_i \in \mathbb{R}^{|\mathcal{O}|}$ and multi-variate normal distribution $\mathcal{N}(\boldsymbol{o}_i|\boldsymbol{h}_o, \Sigma)$ into our relaxation, our relaxed posterior distribution of neural architectures can be framed as

$$\widehat{p}_{\boldsymbol{\alpha}}(\mathcal{A}) = \prod_{i=2}^{N-2} 1/Z_i \sum_{o \in \mathcal{O}} p_{\boldsymbol{\alpha}_i}(o) \mathcal{N}(\boldsymbol{o}_i|\boldsymbol{h}_o, \Sigma) \,, \tag{5.8}$$

where Z_i denotes the normalization constant. Given the sampled particle $\boldsymbol{x}^* = (\cdots \boldsymbol{o}_i^* \cdots)$ in SVGD-RD, the final selected architecture can then be derived using the determination of each selected operation o_i^*, i.e.,

$$o_i^* = \arg\min_{o \in \mathcal{O}} \|\boldsymbol{o}_i^* - \boldsymbol{h}_o\|_2 \,. \tag{5.9}$$

Optimization details. Since Liu and Wang (2016) have demonstrated that SVGD is able to handle unnormalized target distributions, the normalization constant in (5.8) can then be ignored in our SVGD-RD algorithm. In practice, the covariance matrix Σ in (5.8) is set to an identity matrix scaled by $|\mathcal{O}|$. Besides, the parameter δ is optimized as a hyperparameter via grid search or Bayesian Optimization (Snoek et al., 2012) within the range of $[-2, 1]$ in practice. To obtain well-performing particles in our SVGD-RD algorithms efficiently, we apply SGD using the gradient provided in Sec. 5.2.3.2 with a *radial basis function* (RBF) kernel on randomly initialized particles for L=1000 iterations under a learning rate of 0.1 and a momentum of 0.9.

5.3.5 Evaluation on Benchmark Datasets

Evaluation on CIFAR-10/100. We apply the same constructions in DARTS (Liu et al., 2019) for our final performance evaluation on CIFAR-10/100: The final selected architectures consist of 20 cells, and 18 of them are identical normal cells, with the rest being the identical reduction cell. An auxiliary tower with a weight of 4 is located at the 13-th cell of the final selected architectures. The final selected architecture is then trained using stochastic gradient descent (SGD) for 600 epochs with a learning rate cosine scheduled from 0.025 to 0, momentum 0.9, weight decay 3×10^{-4}, batch size 96 and initial channels 36. Cutout (Devries and Taylor, 2017), and a scheduled DropPath, i.e., linearly decayed from 0.2 to 0, are employed to achieve SOTA generalization performance.

Evaluation on ImageNet. Following (Liu et al., 2019), the architectures evaluated on ImageNet consist of 14 cells (12 identical normal cells and 2 identical reduction cells). To meet the requirement of evaluation under the mobile setting (less than 600M multiply-add operations), the number of initial channels for final selected architectures are conventionally set to 44. We adopt the training enhancements in Liu et al. (2019); Chen et al. (2019); Chen and Hsieh (2020), including an auxiliary tower of weight 0.4 and label smoothing. Following P-DARTS Chen et al. (2019) and SDARTS-ADV Chen and Hsieh (2020), we train the selected architectures from scratch for 250 epochs using a batch size of 1024 on 8 GPUs, SGD optimizer with a momentum of 0.9 and a weight decay of 3×10^{-5}. The learning rate applied in this training is warmed up to 0.5 for the first 5 epochs and then decreased to zero linearly.

5.3.6 Adversarial Defense

Adversarial attack intends to find a small change for each input such that this input with its corresponding small change will be misclassified by a model. As ensemble is known to be a possible defense against such adversarial attacks (Strauss et al., 2017), we also examine the effectiveness of our NESBS algorithms by comparing the model robustness achieved by our algorithms to other ensemble and ensemble search algorithms under various benchmark adversarial attacks. To the best of our knowledge, we are the first to examine the advantages of ensemble search algorithms in defending against adversarial attacks.

In this experiment, two processes are required, i.e., *attack* and *defense*. The *attack* process is a typical white-box attack scenario: Only a single model (randomly sampled from an ensemble) is attacked by an attacker, and this process will be repeated for n rounds given an ensemble of size n in order to accurately measure the improvement of model robustness induced by an ensemble. In each round, a different model from this ensemble is selected to be attacked. The *defense* process is then applied using neural network ensembles, i.e., neural network ensembles will make predictions based on those perturbed images produced by the aforementioned attacker. Corresponding to the *attack* process, we also need to repeat this defense process for n rounds. In fact, such an adversarial defense setting is reasonably practical when only a single model from an ensemble is required to be publicly available for model producers.

We apply the following attacks in this experiment: The *Fast Gradient Signed Method* (FGSM) attack Goodfellow et al. (2015), the *Projected Gradient Descent* (PGD) attack Madry et al. (2018), the *Carlini Wagner* (CW) attack Carlini and Wagner (2017) and also the AutoAttack (Croce

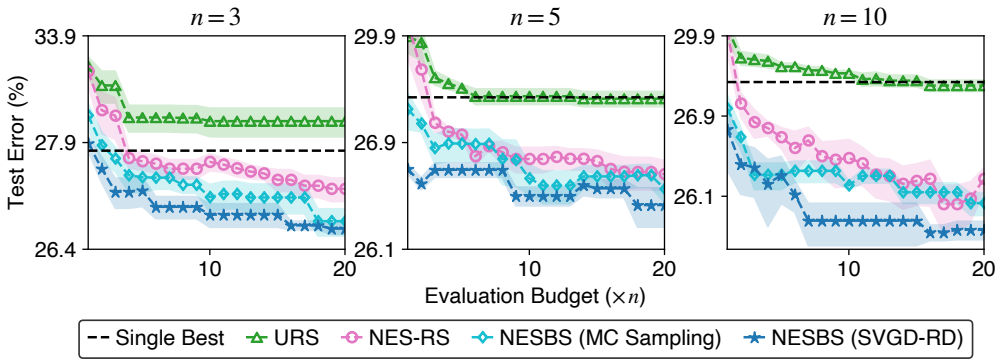


Figure 5.3: The comparison of search effectiveness (test error of ensembles in the y-axis) and efficiency (evaluation budget in the x-axis) for different ensemble search algorithms under varying ensemble size n. The single best baseline refers to the single best architecture achieving the lowest test error in the search space. The y-axis is shown in log-scale to ease visualization. Note that the test error for each algorithm is reported with the mean and standard error of five independent trials.

and Hein, 2020). In both the FGSM attack and the PGD attack, we impose a L_∞ norm constrain of 0.01. The step size and the number of iterations in the PGD attack are set to 0.008 and 40, respectively. We adopt the same configurations of the CW attack under a L_2 norm constrain in (Carlini and Wagner, 2017): We set the confidence constant, the range of constant c, the number of binary search steps, and the maximum number of optimization steps to 0, $[0.001, 10]$, 3, and 50, respectively; we then adopt Adam (Kingma and Ba, 2015) optimizer with learning rate 0.01 and $\beta_1 = 0.9$, $\beta_2 = 0.999$ in its search process. Besides, we adopt the same configuration of AutoAttack from (Croce and Hein, 2020).

5.4 Experiments

5.4.1 Search Effectiveness and Efficiency

As justified in Sec 5.2.3, both our MC Sampling and SVGD-RD algorithms can sample neural architectures with competitive single-model

performances and diverse model predictions, which are known to be the criteria for well-performing ensembles (Zhou, 2012). To further demonstrate that our algorithms are capable of selecting well-performing ensembles effectively and efficiently based on this sampling property, we compare our NESBS algorithm, including NESBS (MC Sampling) and NESBS (SVGD-RD), with the following ensemble search baselines on CIFAR-10 (Krizhevsky et al., 2009) in the DARTS (Liu et al., 2019) search space: (a) Uniform random sampling which we refer to as URS, and (b) NES-RS (Zaidi et al., 2020). That is, we only replace the Bayesian sampling in our NESBS algorithm with these two different sampling/selection algorithms in this experiment and we keep using the model parameters inherited from a supernet to estimate the single-model and ensemble performances of architectures (including the test errors). The detailed experimental settings are in Sec. 5.3.

Figure 5.3 illustrates the search results. Note that both NES-RS and our NESBS are able to achieve lower test errors than the single best-performing architecture in the search space. These results therefore demonstrate that these two ensemble search algorithms are indeed capable of achieving improved performance over conventional NAS algorithms that select only one single architecture from the search space. More importantly, given the same evaluation budgets, our NESBS algorithm consistently achieves lower test errors than URS and NES-RS, indicating the superior search effectiveness achieved by our NESBS algorithm. Meanwhile, our NESBS algorithm requires fewer evaluation budgets than URS and NES-RS to achieve comparable test errors, which also suggests that our algorithm is more efficient than URS and NES-RS. Interestingly, compared with MC Sampling, SVGD-RD can consistently produce improved search effectiveness and efficiency, which likely results

from its controllable trade-off between the single-model performances and the diverse model predictions as justified in Sec. 5.2.3. Overall, these results have well justified the effectiveness and efficiency of our NESBS algorithm.

5.4.2 Search in NAS-Bench-201

To verify the effectiveness and efficiency of our NESBS algorithm, we firstly compare it with other well-known NAS and ensemble (search) algorithms in NAS-Bench-201 (Dong and Yang, 2020). Table 5.1 summarizes the results. Table 5.1 shows that ensemble (search) algorithms, including our NESBS, consistently achieve improved generalization performance over conventional NAS algorithms. This is because ensemble (search) algorithms will select neural network ensembles whereas NAS algorithms will select only one single architecture. Moreover, it has been widely verified that model ensembles generally outperform a single machine learning model in practice (Zhou, 2012). In addition, our NESBS algorithm outperforms other ensemble (search) baseline (i.e., DeepEns and NES-RS), especially on large-scale datasets (i.e., CIFAR-100 (Krizhevsky et al., 2009) and ImageNet-16-200 (Chrabaszcz et al., 2017)) while incurring less search costs than NES-RS, which thus implies the superior performance of our NESBS over these ensemble (search) baselines. Even on a small-scale dataset (i.e., CIFAR-10), our NESBS can also achieve comparable search results to DeepEns and NES-RS. Interestingly, our NESBS algorithm is even able to incur reduced search costs than conventional NAS algorithms. This is likely because more training epochs have been used in these NAS algorithms, whereas a small number of training epochs can already contribute to well-performing results for our NESBS algorithm.

Table 5.1: Comparison of architectures selected by different NAS and ensemble (search) algorithms in NAS-Bench-201 with ensemble size $n = 3$. Test errors are reported with the mean and standard error of three independent trials and our search costs are evaluated on a single Nvidia 1080Ti GPU. Results marked by † are reported by Dong and Yang (2020).

Architecture(s)	Test Error (%)			Search Cost (GPU Hours)
	CIFAR-10	CIFAR-100	ImageNet-16-200	
		Manual design		
ResNet† (He et al., 2016)	6.03	29.14	56.37	-
		NAS algorithms		
ENAS† (Pham et al., 2018)	45.70±0.00	84.39±0.00	83.68±0.00	3.7
DARTS† (2nd) (Liu et al., 2019)	45.70±0.00	84.39±0.00	83.68±0.00	8.3
GDAS† (Dong and Yang, 2019b)	6.49±0.13	29.39±0.26	58.16±0.90	8.0
SETN† (Dong and Yang, 2019a)	13.81±4.63	43.13±7.77	68.10±4.07	8.6
RSPS† (Li and Talwalkar, 2019b)	12.34±1.69	41.67±4.34	68.86±3.88	2.1
		Ensemble (search) algorithms		
DeepEns (Lakshminarayanan et al., 2017)	5.75	25.27	54.70	-
NES-RS (Zaidi et al., 2020)	5.83±0.33	25.58±0.84	54.34±1.67	5.1
		Our ensemble search algorithm		
NESBS (MC Sampling)	5.76±0.25	25.39±0.69	**53.47**±1.75	**1.1**
NESBS (SVGD-RD)	5.92±0.07	**25.00**±0.17	**52.68**±0.35	**1.2**

5.4.3 Search in The DARTS Search Space

We further demonstrate the superior search effectiveness and efficiency of our NESBS by comparing it with other NAS and ensemble (search) baselines in a larger search space (i.e., DARTS (Liu et al., 2019) search space) using both classification and adversarial defense tasks on CIFAR-10/100 or ImageNet (Deng et al., 2009). We follow Sec. 5.3.5 to evaluate the final neural network ensembles selected by our NESBS algorithm with ensemble size $n = 3$, $T = 5$, and optimization details in Sec. 5.3.

Ensemble for classification. Table 5.2 summarizes the comparison of classification performances on CIFAR-10/100. Similar to the results in Sec. 5.4.2, ensemble (search) algorithms, including our NESBS, are generally able to achieve improved generalization performances over conventional NAS algorithms, which thus justifies the essence of ensemble (search) algorithms for improved performance. Notably, even compared with other ensemble baselines such as MC DropPath (i.e., de-

Table 5.2: Comparison of different image classifiers on CIFAR-10/100. Results of MC DropPath are from a drop probability of 0.01 and our search costs are evaluated on Nvidia 1080Ti.

Architecture(s)	Test Error (%)		Params (M)		Search Cost (GPU Days)	Search Method
	C10	C100	C10	C100		
NAS algorithms						
NASNet-A (Zoph et al., 2018)	2.65	-	3.3	-	2000	RL
AmoebaNet-A (Real et al., 2019b)	3.34	18.93	3.2	3.1	3150	evolution
PNAS (Liu et al., 2018)	3.41	19.53	3.2	3.2	225	SMBO
ENAS (Pham et al., 2018)	2.89	19.43	4.6	4.6	0.5	RL
DARTS (Liu et al., 2019)	2.76	17.54	3.3	3.4	1	gradient
GDAS (Dong and Yang, 2019b)	2.93	18.38	3.4	3.4	0.3	gradient
P-DARTS (Chen et al., 2019)	2.50	-	3.4	-	0.3	gradient
DARTS- (avg) (Chu et al., 2020)	2.59	17.51	3.5	3.3	0.4	gradient
SDARTS-ADV (Chen and Hsieh, 2020)	2.61	-	3.3	-	1.3	gradient
Ensemble (search) algorithms						
MC DropPath (ENAS)	2.88	16.83	3.8‡	3.9‡	-	-
DeepEns (ENAS)	2.49	15.04	3.8‡	3.9‡	-	-
DeepEns (DARTS)	2.42	14.56	3.3‡	3.4‡	-	-
NES-RS♯ (Zaidi et al., 2020)	2.50	15.24	3.0‡	3.1‡	0.7	greedy
Our ensemble search algorithm						
NESBS (MC Sampling)	**2.41**	14.70	3.8‡	3.9‡	**0.2**	sampling
NESBS (SVGD-RD)	**2.36**	**14.55**	3.7‡	3.8‡	**0.2**	sampling

‡ Reported as the averaged parameter size of the architectures in a neural network ensemble.
♯ Obtained from a pool of size 50, in which every architecture is uniformly randomly sampled from the DARTS search spaces and then trained independently for 50 epochs following the evaluation settings in Sec. 5.3.5.

Table 5.3: Comparison of image classifiers on ImageNet. The ensemble size is set to $n = 3$ for NES-RS and NESBS.

Architecture(s)	Test Error (%)		Params (M)	+× (M)
	Top-1	Top-5		
NAS algorithms				
NASNet-A	26.0	8.4	5.3	564
AmoebaNet-A	25.5	8.0	5.1	555
PNAS	25.8	8.1	5.1	588
DARTS	26.7	8.7	4.7	574
GDAS	26.0	8.5	5.3	581
P-DARTS	24.4	7.4	4.9	557
SDARTS-ADV	25.2	7.8	5.4	594
Ensemble (search) algorithm				
NES-RS	23.4	6.8	3.9	432
Our ensemble search algorithm				
NESBS (MC Sampling)	**22.3**	**6.2**	4.6	522
NESBS (SVGD-RD)	**22.3**	**6.1**	4.9	562

veloped following Monte Carlo Dropout (Gal and Ghahramani, 2016)) and DeepEns, our NESBS is still able to achieve improved performances. Since these ensemble baselines are orthogonal to our NESBS, they can

be integrated into our NESBS for further performance improvement in real-world applications. More importantly, our algorithm outperforms NES-RS by achieving both improved search effectiveness (lowest test errors) and efficiency (lowest search costs). Furthermore, our NESBS even incurs comparable search costs compared with the most efficient NAS algorithms (e.g., GDAS, P-DARTS), which also highlights the efficiency of our NESBS. Similar results on ImageNet can be achieved by our NESBS as shown in Table 5.3. [3]

Ensemble for adversarial defense. Ensemble methods have already been shown to be an essential and effective defense mechanism against adversarial attacks (Strauss et al., 2017). Specifically, an adversarial attacker can only use *a single model* randomly sampled from an ensemble to generate the adversarial examples, whereas the ensemble method defends against adversarial attacks (i.e., makes its predictions) using *all models* in this ensemble. Ensemble methods can defend against the adversarial attacks in such a setting because the generated adversarial examples using only one single model are unlikely to fool all models in an ensemble. More details are provided in Sec. 5.3.6. Table 5.4 summarizes the comparison of adversarial defense among ensemble (search) algorithms on CIFAR-10/100 under different white-box adversarial attacks, including the *Fast Gradient Signed Method* (FGSM) attack Goodfellow et al. (2015), the *Projected Gradient Descent* (PGD) attack Madry et al. (2018), the *Carlini Wagner* (CW) attack Carlini and Wagner (2017), and the AutoAttack (Croce and Hein, 2020). Table 5.4 shows that ensemble (search) algorithms are indeed able to significantly improve the performance of

[3] Following the convention of NAS and ensemble search algorithms in Table 5.3, the ensembles selected by our NESBS are also searched on CIFAR-10 and then transferred to ImageNet.

Table 5.4: Comparison of adversarial defense among different ensemble (search) algorithms on CIFAR-10/100 under white-box adversarial attacks. The *Attack* and *Defense* columns denote the test *accuracy* under the attack using a single model randomly sampled from an ensemble and the defense using the whole ensemble, respectively. Each result reports the mean and standard deviation of test accuracies for 3 rounds of the attack-defense process with an ensemble size of $n = 3$.

Method	**FGSM**		**PGD-40**		**CW**		**AutoAttack**	
	Attack (%)	Defense (%)	Attack (%)	Defense (%)	Attack (%)	Defense (%)	Attack (%)	Defense (%)
				On CIFAR-10 Dataset				
DeepEns	-	-	-	-	-	-	-	-
↪ RobNet-free	66.62±0.32	85.25±0.39	41.81±0.80	77.48±0.67	5.74±1.41	86.53±0.50	21.35±0.33	45.51±0.15
↪ ENAS	77.85±0.58	87.94±0.21	59.51±1.13	86.57±0.15	31.36±1.20	85.20±0.77	31.71±0.72	50.96±0.07
↪ DARTS	76.79±0.80	88.21±0.14	57.71±1.65	82.02±0.10	26.90±1.37	82.46±0.35	29.97±1.17	49.67±0.14
NES-RS	79.19±1.39	89.32±0.27	65.59±2.11	85.22±0.41	37.20±4.62	86.75±0.88	35.00±1.15	53.80±0.14
NESBS (MC Sampling)	78.75±1.29	89.15±0.08	63.60±1.87	85.35±0.31	**37.71**±1.97	**86.86**±0.66	**36.02**±0.64	**56.90**±0.17
NESBS (SVGD-RD)	79.12±0.61	**89.86**±0.33	65.53±1.56	85.37±0.38	**38.27**±1.27	86.00±1.10	**37.55**±0.68	**57.15**±0.20
				On CIFAR-100 Dataset				
DeepEns	-	-	-	-	-	-	-	-
↪ RobNet-free	36.47±0.25	61.39±0.30	18.18±0.47	52.61±0.13	2.36±0.13	69.44±0.04	7.31±0.35	24.56±0.33
↪ ENAS	46.40±0.37	64.94±0.27	28.87±0.27	56.79±0.25	9.60±0.30	69.43±0.44	11.53±0.47	27.01±0.27
↪ DARTS	46.98±0.57	65.38±0.23	28.78±0.74	57.10±0.04	9.73±0.43	70.15±0.29	11.20±0.40	26.86±0.36
NES-RS	47.10±1.46	65.33±0.36	30.68±1.66	58.80±0.80	9.96±1.45	70.24±0.33	12.01±0.93	27.49±0.34
NESBS (MC Sampling)	**50.69**±1.58	**67.63**±0.05	**33.37**±0.42	**60.36**±0.62	**15.64**±2.83	**71.25**±1.27	**13.11**±1.16	**29.87**±1.17
NESBS (SVGD-RD)	**51.47**±0.40	**66.66**±0.13	**35.02**±0.37	**59.96**±0.18	**16.72**±0.61	**69.88**±0.16	**14.62**±0.55	**31.07**±0.33

adversarial defense, i.e., the test accuracies in the *Defense* column are consistently higher than the ones in *Attack* column. More importantly, even under different white-box adversarial attacks, our NESBS algorithm can generally achieve improved defense performances (i.e., higher test accuracy in the *Defense* columns) than other baselines including DeepEns and NES-RS. These results thus further support the effectiveness of our NESBS over existing ensemble (search) algorithms. Besides, even regarding the adversarial robustness of the single models in an ensemble, the architectures selected by our NESBS are also more advanced (i.e., by achieving higher test accuracy in the *Attack* columns) than well-known architectures such as RobNet (Guo et al., 2020) and DARTS.

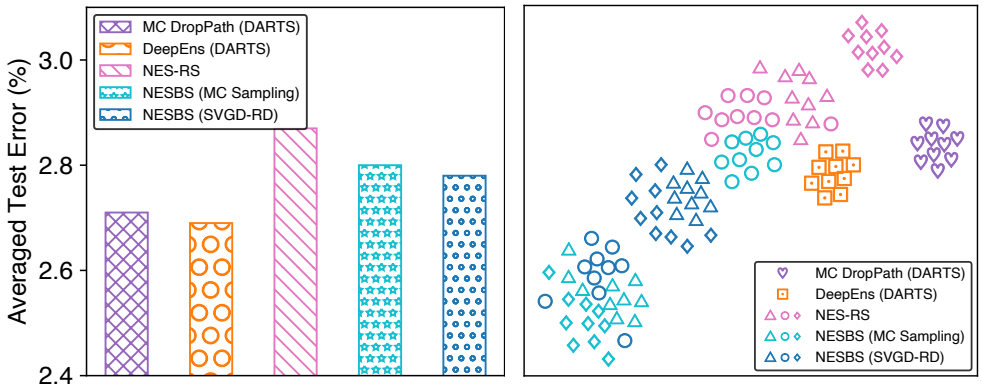
Table 5.5: Quantitative comparison of the single-model performances (measured by ATE (%), smaller is better) and the diversity of model predictions (measured by PPD (%), larger is better) achieved by different ensemble (search) algorithms with an ensemble size of 3 on CIFAR-10/100.

Method	**C10**		**C100**	
	ATE	PPD	ATE	PPD
MC DropPath (DARTS)	2.71	0.39	16.68	2.63
DeepEns (DARTS)	**2.69**	2.08	**16.18**	12.45
NES-RS	2.87	2.29	17.20	**14.14**
NESBS (MC Sampling)	2.80	**2.57**	16.70	13.84
NESBS (SVGD-RD)	2.78	2.27	16.50	13.16

5.4.4 Single-model Performances and Diverse Model Predictions

We demonstrate that the effectiveness of our NESBS results from its ability to achieve a good trade-off between the single-model performances and the diversity of model predictions. We firstly quantitatively compare the single-model performances (measured by the *averaged test error* (ATE) of the models in an ensemble) and the diversity of model predictions (measured by the *pairwise predictive disagreement* (PPD) of an ensemble (Fort et al., 2019)) achieved by different ensemble (search) algorithms on CIFAR-10/100. We further qualitatively visualize their single-model performances and diverse model predictions using a histogram of the ATE of the models in their ensembles and a t-SNE (Van der Maaten and Hinton, 2008) plot of their model predictions, respectively.

Table 5.5 and Fig. 5.4 present the results of our quantitative and qualitative comparisons, respectively. Compared with the ensemble baselines of MC DropPath and DeepEns, our NESBS is capable of enjoying a larger diversity of model predictions while preserving competitive single-model performances. Meanwhile, compared with the ensemble search baselines of NES-RS, our algorithm can achieve improved single-model perfor-



(a) Single-model performances (b) Diverse predictions

Figure 5.4: Qualitative comparison of (a) the single-model performances and (b) the diverse model predictions achieved by different ensemble (search) algorithms with an ensemble size of $n = 3$ on CIFAR-10. Each architecture in (b) is independently evaluated for ten times to visualize their model predictions, which follows from DeepEns.

mances while maintaining comparably diverse model predictions. These results suggest that our NESBS is able to select ensembles achieving a better trade-off between the single-model performances and the diversity of model predictions among these baselines, which is known to be an important criterion for well-performing ensembles (Zhou, 2012). Thus, Table 5.5 and Fig. 5.4 provide empirical justifications for the improved effectiveness of NESBS.

5.5 More Results

5.5.1 Ensemble Performance Estimation

As shown in Sec. 5.2.1, we apply the model parameters inherited from a trained supernet to estimate the performance of candidate architectures as well as their ensembles in our NESBS algorithm. We therefore use the following three metrics to measure the effectiveness of such estimation in the DARTS search space: the Spearman's rank order coefficient between

Table 5.6: The correlation between the estimated and true performances of candidate architectures and their ensembles in the DARTS search space on CIFAR-10.

Metric	$n = 1$	$n = 3$	$n = 5$	$n = 7$
Spearman	0.65	0.33	0.40	−0.12
Pearson	0.82	0.45	0.45	−0.16
Agreement-30%	33%	20%	31%	25%

the estimated and true performances, the Pearson correlation coefficient between the estimated and true performances, and the percentage of architectures achieving both Top-k estimated performance and Top-k true performances (named the Agreement-k). Since the evaluation of the true performances is prohibitively costly, we randomly sample 10 architectures of diverse estimated performances from the DARTS search space for this experiment. Notably, based on these 10 architectures, there are hundreds of possible ensembles under the ensemble size of 3, 5, 7, which we believe is sufficiently large to validate the effectiveness of our performance estimations. To obtain the true performance of candidate architectures as well as their ensembles, we train these architectures independently for 100 epochs following the settings in Sec. 5.3.5.

Table 5.6 summarizes the results. Notably, the estimated and true performances are shown to be positively correlated in the case of n=1, 3, 5 by achieving relatively high Spearman and Pearson coefficients as well as a high agreement in these cases. Although the coefficients are low when the ensemble size is larger (i.e., n=7), the estimated and true performances are still capable of achieving a reasonably good agreement in this case. Based on these results, we argue that our estimated ensemble performance is informative and effective for our ensemble search. This effectiveness can also be supported by the competitive search results achieved by our NESBS in Sec. 5.4.3.

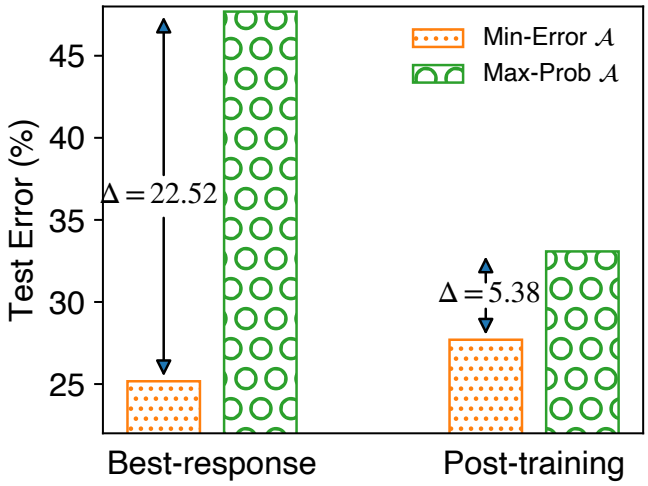


Figure 5.5: The comparison of performance discrepancy with the post-training and best-response posterior distribution on CIFAR-10. This performance discrepancy is measured by the gap of test error between the best-performing architecture (i.e., the architecture with the smallest test error) and the maximal-probability architecture (i.e., the architecture with the largest probability in the corresponding posterior distribution) in the DARTS search space.

5.5.2 Post-training vs. Best-response Posterior Distribution

To examine the advantages of our post-training posterior distribution, we compare it with its best-response counterpart applied in (Dong and Yang, 2019b; Xie et al., 2019b). While our post-training posterior distribution is obtained *after* the model training of the supernet, the best-response posterior distribution is updated *during* the model training of the supernet. We refer to (Dong and Yang, 2019b; Xie et al., 2019b) for more details about this best-response posterior distribution. We follow the optimization details in Sec. 5.3.2 and 5.3.3 to obtain these two posterior distributions.

More accurate characterization of single-model performances using post-training posterior distribution. We firstly compare the charac-

terization of single-mode performance using these two posterior distributions by examining the performance discrepancy between their best-performing architecture (i.e., the architecture achieving the smallest test error) and maximal-probability architecture (i.e., the architecture achieving the largest probability in the corresponding posterior distribution) in the search space. In this experiment, the performance discrepancy is measured by the gap of test error achieved by the best-performing architecture and the maximal-probability architecture using the model parameters inherited from the supernet.

Figure 5.5 illustrates the comparison. The results show that our post-training posterior distribution enjoys a smaller performance discrepancy, suggesting that our post-training posterior distribution is able to provide a more accurate characterization of the single-model performances. Interestingly, the best-response counterpart contributes to the best-performing architecture with a lower test error than our post-training posterior distribution, which should result from the Matthew Effect as justified in (Hong et al., 2020). Specifically, well-performing architectures contribute to the frequent selections of these architectures for their model training during the optimization of the best-response posterior distribution. This will finally result in unfair model training in the search space and therefore the inaccurate characterization of single-model performances. Notably, we need a more accurate characterization of single-mode performance in this work, as shown in Sec. 5.2.2. Therefore, our post-training posterior distribution should be more suitable than its best-response counterpart in our ensemble search.

Improved performance of selected ensembles using post-training posterior distribution. We then compare the final ensemble test perfor-

Table 5.7: The comparison of true ensemble test error (%) on CIFAR-10 achieved by our NESBS algorithm using the post-training posterior distribution and its best-response counterpart with an ensemble size of n=3. We use Δ to denote the improved generalization performance achieved by our post-training posterior distribution.

Method	**Best-response**	**Post-training**
NESBS (MC Sampling)	4.74	$\mathbf{4.54}_{\Delta=0.20}$
NESBS (SVGD-RD)	4.81	$\mathbf{4.48}_{\Delta=0.33}$

mance achieved by our NESBS algorithm using the post-training posterior distribution and its best-response counterpart on CIFAR-10 with the ensemble size of $n = 3$. To obtain the final ensemble performance, we train each architecture in an ensemble for 100 epochs following the settings in Sec. 5.3.5. Table 5.7 summarizes the results. Notably, our post-training posterior distribution is shown to be capable of contributing to an improved ensemble performance than its best-response counterpart, which further demonstrates the advantages of applying the post-training posterior distribution in our ensemble search.

5.5.3 The Advantages of Controllable Diversity in SVGD-RD

To examine the advantages of controllable diversity in our SVGD-RD, we firstly compare the search effectiveness and efficiency achieved by our NESBS (MC Sampling) and NESBS (SVGD-RD) algorithm with varying softmax temperature τ (appeared in (5.6)). A larger temperature τ will lead to a flatter posterior distribution and hence degenerate its capability of characterizing single-model performances of neural architectures as indicated in (5.6). We use these posterior distributions with varying temperature τ to simulate the possible posterior distributions we may obtain in practice. Figure 5.6 illustrates the comparison on CIFAR-10 in

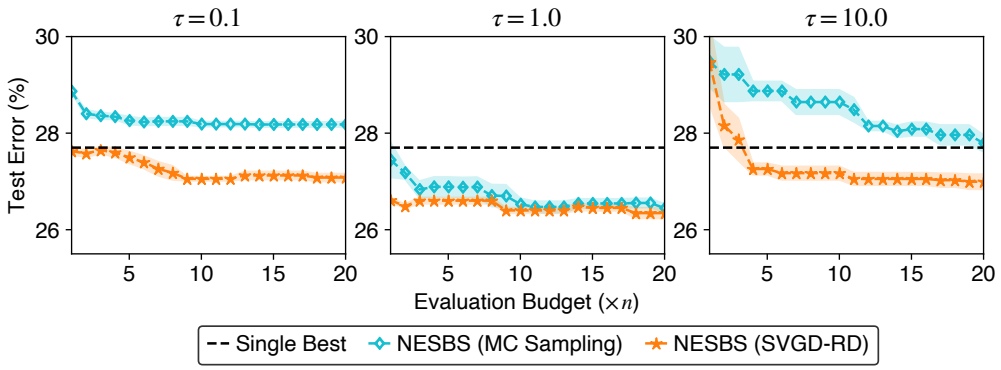


Figure 5.6: The comparison of search effectiveness (test error of ensembles in the y-axis) and efficiency (evaluation budget in the x-axis) between our NESBS (MC Sampling) and NESBS (SVGD-RD) algorithm under varying softmax temperature τ. The single best baseline refers to the single best architecture achieving the lowest test error in the search space. Each test error is reported with the mean and standard error of five independent trials.

the DARTS search space with an ensemble size of $n = 5$. Notably, our NESBS (SVGD-RD) with controllable diversity can consistently achieve improved search effectiveness and efficiency than our NESBS (MC Sampling). Interestingly, this improvement becomes larger in the case of $\tau = 0.1, 10.0$, which should be the consequences of a bad exploration and exploitation achieved by our NESBS (MC Sampling), respectively. These results therefore suggest that the controllable diversity in our SVGD-RD generally can lead to improved search effectiveness and efficiency than our NESBS (MC Sampling).

We further provide the comparison of ensemble test error achieved by our SVGD-RD with varying δ under different softmax temperature τ in Figure 5.7. Notably, when the posterior distribution tends to be flatter (i.e., $\tau = 10$), a smaller δ is preferred by our SVGD-RD in order to sample architectures with better single-model performances while maintaining the compelling diverse model predictions. Meanwhile, when this posterior distribution tends to be sharper (i.e., $\tau = 0.1$), a larger δ is preferred by our SVGD-RD in order to sample architectures with

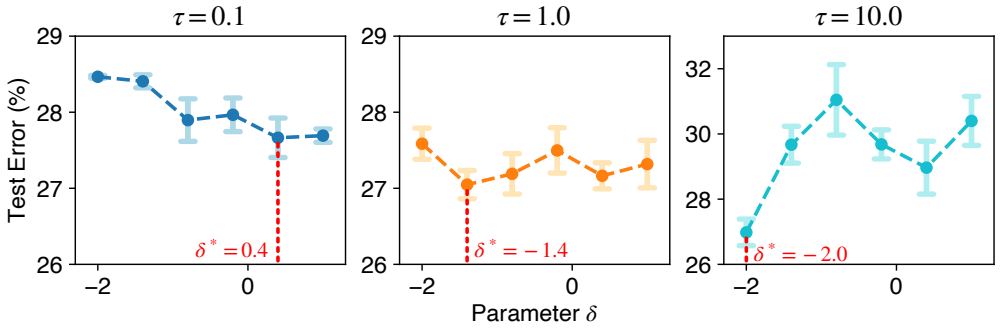


Figure 5.7: The comparison of ensemble test error achieved by our NESBS (SVGD-RD) algorithm with varying δ under different softmax temperature τ given an ensemble size of $n = 5$. We use δ^* to denote the optimal δ we obtained in our SVGD-RD algorithm under different temperature τ. The test error for each δ is reported with the mean and standard error of five independent trials.

a larger diverse model predictions while preserving the competitive single-model performances. Based on this controllable diversity and hence the controllable trade-off between the single-model performances and the diverse model predictions, our SVGD-RD is thus capable of achieving comparable performances under varying τ, which usually improve over our NESBS (MC Sampling) by comparing them with the results in Figure 5.6. These results further validate the advantages of the controllable diversity in our SVGD-RD.

5.6 Conclusion

This work presents a novel neural ensemble search algorithms, called NESBS, that can effectively and efficiently select well-performing neural network ensembles with diverse architectures from a NAS search space. Our extensive experiments have shown that NESBS is able to achieve improved performances while preserving a comparable search cost compared with conventional NAS algorithms. Moreover, even compared with other ensemble (search) baselines (e.g., DeepEns and NES-RS), our

NESBS is also capable of enjoying boosted search effectiveness and efficiency, which further suggests the superior performance of our NESBS in practice.

Chapter 6

NASI: Label- and Data-agnostic Neural Architecture Search at Initialization

This chapter is based on the following work published at ICLR 2022: Shu, Y., Cai, S., Dai, Z., Ooi, B. C., & Low, B. K. H. (2022). NASI: Label- and Data-agnostic Neural Architecture Search at Initialization. In *Proc. ICLR-22*.

6.1 Introduction

The past decade has witnessed the wide success of *deep neural networks* (DNNs) in computer vision and natural language processing. These DNNs, e.g., VGG (Simonyan and Zisserman, 2015), ResNet (He et al., 2016), and MobileNet (Howard et al., 2017), are typically handcrafted by human experts with considerable trials and errors. The human efforts devoting to the design of these DNNs are, however, not affordable nor scalable due to an increasing demand of customizing DNNs for different

tasks. To reduce such human efforts, *Neural Architecture Search* (NAS) (Zoph and Le, 2017) has recently been introduced to automate the design of DNNs. As summarized in (Elsken et al., 2019a), NAS conventionally consists of a search space, a search algorithm, and a performance evaluation. Specifically, the search algorithm aims to select the best-performing neural architecture from the search space based on its evaluated performance via performance evaluation. In the literature, various search algorithms (Luo et al., 2018a; Zoph et al., 2018; Real et al., 2019b) have been proposed to search for architectures with comparable or even better performance than the handcrafted ones.

However, these NAS algorithms are inefficient due to the requirement of model training for numerous candidate architectures during the search process. To improve the search inefficiency, one-shot NAS algorithms (Dong and Yang, 2019b; Pham et al., 2018; Liu et al., 2019; Xie et al., 2019b) have trained a single one-shot architecture and then evaluated the performance of candidate architectures with model parameters inherited from this fine-tuned one-shot architecture. So, these algorithms can considerably reduce the cost of model training, but still require the training of the one-shot architecture. This naturally leads to the question *whether NAS is realizable at initialization such that model training can be completely avoided during the search process?* To the best of our knowledge, only a few efforts to date have been devoted to developing NAS algorithms without model training empirically (Mellor et al., 2020a; Park et al., 2020; Abdelfattah et al., 2021; Chen et al., 2021).

This work presents a novel NAS algorithm called *NAS at Initialization* (NASI) that can completely avoid model training to boost search efficiency. To achieve this, NASI exploits the capability of a *Neural Tangent Kernel* (NTK) (Jacot et al., 2018; Lee et al., 2019a) in being able to formally char-

acterize the performance of infinite-wide DNNs at initialization, hence allowing the performance of candidate architectures to be estimated and realizing NAS at initialization. Specifically, given the estimated performance of candidate architectures by NTK, NAS can be reformulated into an optimization problem without model training (Sec. 6.2.1). However, NTK is prohibitively costly to evaluate. Fortunately, we can approximate it[1] with a similar form to gradient flow (Wang et al., 2020b) (Sec. 6.2.2). This results in a reformulated NAS problem that can be solved efficiently by a gradient-based algorithm via additional relaxation with Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017) (Sec. 6.2.3). Interestingly, NASI is shown to be *label-* and *data-agnostic* under mild conditions, which thus implies the transferability of architectures selected by NASI over different datasets (Sec. 6.3).

We will firstly empirically demonstrate the improved search efficiency and the competitive search effectiveness achieved by NASI in NAS-Bench-1Shot1 (Zela et al., 2020b) (Sec. 6.5.1). Compared with other NAS algorithms, NASI incurs the smallest search cost while preserving the competitive performance of its selected architectures. Meanwhile, the architectures selected by NASI from the DARTS (Liu et al., 2019) search space over CIFAR-10 consistently enjoy the competitive or even outperformed performance when evaluated on different benchmark datasets, e.g., CIFAR-10/100 and ImageNet (Sec. 6.5.3), indicating the guaranteed transferability of architectures selected by our NASI. In Sec. 6.5.4, NASI is further demonstrated to be able to select well-performing architectures on CIFAR-10 even with randomly generated labels or data, which strongly supports the *label-* and *data-agnostic* search and also the guaranteed transferability achieved by our NASI.

[1] More precisely, we approximate the trace norm of NTK.

6.2 Neural Architecture Search at Initialization

6.2.1 Reformulating NAS via NTK

Given a loss function $\mathcal{L}$ and model parameters $\theta(\mathcal{A})$ of architecture $\mathcal{A}$, we denote the training and validation loss as $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$, respectively. NAS is conventionally formulated as a bi-level optimization problem (Liu et al., 2019):

$$\begin{aligned} &\min_{\mathcal{A}} \mathcal{L}_{\text{val}}(\theta^*(\mathcal{A}); \mathcal{A}) \\ &\text{s.t. } \theta^*(\mathcal{A}) \triangleq \arg\min_{\theta(\mathcal{A})} \mathcal{L}_{\text{train}}(\theta(\mathcal{A}); \mathcal{A}) \, . \end{aligned} \tag{6.1}$$

Notably, model training is required to evaluate the validation performance of each candidate architecture in (6.1). The search efficiency of NAS algorithms (Real et al., 2019b; Zoph et al., 2018) based on (6.1) is thus severely limited by the cost of model training for each candidate architecture. Though recent works (Pham et al., 2018) have considerably reduced this training cost by introducing a one-shot architecture for model parameter sharing, such a one-shot architecture requires training and hence incurs the training cost.

To completely avoid this training cost, we exploit the capability of NTK for characterizing the performance of DNNs at initialization. Specifically, Sec. 2.4 has revealed that the training dynamics of an over-parameterized DNN can be governed by its linearization at initialization. With the MSE loss, the training dynamics of such linearization are further determined by its constant NTK. Therefore, the training dynamics and hence the performance of a DNN can be characterized by the constant NTK of its linearization. However, this constant NTK is computationally costly to evaluate. To this end, we instead characterize the training dynamics (i.e., MSE) of DNNs in Proposition 6.1 using the trace norm

of NTK at initialization, which can be efficiently approximated. For simplicity, we use this MSE loss in our analysis. Other widely adopted loss functions (e.g., cross entropy with softmax) can also be applied, as supported in our experiments. Note that throughout this work, the parameterization and initialization of DNNs follow that of Jacot et al. (2018). For a L-layer DNN, we denote the output dimension of its hidden layers and the last layer as $n_1 = \cdots = n_{L-1} = k$ and $n_L = n$, respectively.

Proposition 6.1. *Suppose that* $\|\boldsymbol{x}\|_2 \leq 1$ *for all* $\boldsymbol{x} \in \mathcal{X}$ *and* $\mathcal{Y} \in [0,1]^{mn}$ *for a given dataset* $(\mathcal{X}, \mathcal{Y})$ *of size* $|\mathcal{X}| = m$*, a given* L*-layer neural architecture* $\mathcal{A}$ *outputs* $\boldsymbol{f}_t \in [0,1]^{mn}$ *as predicted labels of* $\mathcal{Y}$ *with the corresponding MSE loss* $\mathcal{L}_t$*,* $\lambda_{\min}(\boldsymbol{\Theta}_0) > 0$ *for the given NTK* $\boldsymbol{\Theta}_0$ *w.r.t.* $\boldsymbol{f}_t$ *at initialization, and gradient descent (or gradient flow) is applied with learning rate* $\eta < \lambda_{\max}^{-1}(\boldsymbol{\Theta}_0)$*. Then, for any* $t \geq 0$*, there exists a constant* $c_0 > 0$ *such that as* $k \to \infty$*,*

$$\mathcal{L}_t \leq mn^2(1 - \eta\overline{\lambda}(\boldsymbol{\Theta}_0))^q + \epsilon$$

with probability arbitrarily close to 1 *where* q *is set to* $2t$ *if* $t < 0.5$*, and* 1 *otherwise,* $\overline{\lambda}(\boldsymbol{\Theta}_0) \triangleq (mn)^{-1}\sum_{i=1}^{mn}\lambda_i(\boldsymbol{\Theta}_0)$*, and* $\epsilon \triangleq 2c_0\sqrt{n/(mk)}\left(1 + c_0\sqrt{1/k}\right)$*.*

Its proof is in Appendix C.1.3. Proposition 6.1 implies that NAS can be realizable at initialization. Specifically, given a fixed and sufficiently large training budget t, in order to select the best-performing architecture, we can simply minimize the upper bound of $\mathcal{L}_t$ in (6.1) over all the candidate architectures in the search space. Here, $\mathcal{L}_t$ can be applied to approximated $\mathcal{L}_{\text{val}}$ since both strong theoretical (Mohri et al., 2018) and empirical (Hardt et al., 2016) justifications in the literature have shown that training and validation loss are generally highly related. Hence, (6.1)

can be reformulated as

$$\min_{\mathcal{A}} \; mn^2(1-\eta\overline{\lambda}(\mathbf{\Theta}_0(\mathcal{A}))) + \epsilon \quad \text{s.t. } \overline{\lambda}(\mathbf{\Theta}_0(\mathcal{A})) < \eta^{-1}. \tag{6.2}$$

Note that the constraint in (6.2) is derived from the condition $\eta < \lambda_{\max}^{-1}(\mathbf{\Theta}_0(\mathcal{A}))$ in Proposition 6.1, and η and ϵ are typically constants[2] during the search process. Following the definition of trace norm, (6.2) can be further reduced into

$$\max_{\mathcal{A}} \left\|\mathbf{\Theta}_0(\mathcal{A})\right\|_{\text{tr}} \quad \text{s.t. } \left\|\mathbf{\Theta}_0(\mathcal{A})\right\|_{\text{tr}} < mn\eta^{-1} \,. \tag{6.3}$$

Notably, $\mathbf{\Theta}_0(\mathcal{A})$ only relies on the initialization of $\mathcal{A}$. So, no model training is required in optimizing (6.3), which achieves our objective of realizing NAS at initialization.

Furthermore, (6.3) suggests an interesting interpretation of NAS: NAS intends to select architectures with a good trade-off between their model complexity and the optimization behavior in their model training. Particularly, architectures containing more model parameters will usually achieve a larger $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ according to the definition in (2.9), which hence provides an alternative to measuring the complexity of architectures. So, maximizing $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ leads to architectures with large complexity and therefore strong representation power. On the other hand, the complexity of the selected architectures is limited by the constraint in (6.3) to ensure a well-behaved optimization with a large learning rate η in their model training. By combining these two effects, the optimization of (6.3) naturally trades off between the complexity of the selected architectures and the optimization behavior in their model training for the best perfor-

[2]The same learning rate is shared among all candidate architectures in the search space for their model training during the search process in conventional NAS algorithms such as in (Liu et al., 2019).

mance. Sec. 6.6.1 will validate such trade-off. Interestingly, Chen et al. (2021) have revealed a similar insight of NAS to us.

6.2.2 Approximating the Trace Norm of NTK

The optimization of our reformulated NAS in (6.3) requires the evaluation of $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ for each architecture $\mathcal{A}$ in the search space, which can be obtained by

$$\left\|\boldsymbol{\Theta}_0(\mathcal{A})\right\|_{\text{tr}} = \sum_{\boldsymbol{x}\in\mathcal{X}} \left\|\nabla_{\boldsymbol{\theta}_0(\mathcal{A})} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0(\mathcal{A}))\right\|_{\text{F}}^2 , \tag{6.4}$$

where $\|\cdot\|_{\text{F}}$ denotes the Frobenius norm. However, the Frobenius norm of the Jacobian matrix in (6.4) is costly to evaluate. So, we propose to approximate this term. Specifically, given a γ-Lipschitz continuous loss function[3] $\mathcal{L}_{\boldsymbol{x}}$ (i.e., $\left\|\nabla_f \mathcal{L}_{\boldsymbol{x}}\right\|_2 \leq \gamma$ for all $\boldsymbol{x} \in \mathcal{X}$),

$$\gamma^{-1} \left\|\nabla_{\boldsymbol{\theta}_0(\mathcal{A})} \mathcal{L}_{\boldsymbol{x}}\right\|_2 = \gamma^{-1} \left\|\nabla_f \mathcal{L}_{\boldsymbol{x}}^\top \nabla_{\boldsymbol{\theta}_0(\mathcal{A})} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0(\mathcal{A}))\right\|_2 \leq \left\|\nabla_{\boldsymbol{\theta}_0(\mathcal{A})} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0(\mathcal{A}))\right\|_{\text{F}} . \tag{6.5}$$

The Frobenius norm $\left\|\nabla_{\boldsymbol{\theta}_0(\mathcal{A})} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0(\mathcal{A}))\right\|_{\text{F}}$ can therefore be approximated efficiently by its lower bound given in (6.5) through automatic differentiation (Baydin et al., 2017).

Meanwhile, the evaluation of $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ in (6.4) requires iterating over the entire dataset of size m, which incurs $\mathcal{O}(m)$ time. Fortunately, this incurred time can be reduced by parallelization over mini-batches. Let the set $\mathcal{X}_j$ denote the input feature vectors of the j-th randomly sampled

[3]The γ-Lipschitz continuity is satisfied by widely adopted loss functions, as explained in Appendix C.1.3.

mini-batch of size $|\mathcal{X}_j| = b$. By combining (6.4) and (6.5),

$$\left\|\boldsymbol{\Theta}_0(\mathcal{A})\right\|_{\text{tr}} \geq \gamma^{-1} \sum_{\boldsymbol{x}\in\mathcal{X}} \left\|\nabla_{\theta_0(\mathcal{A})}\mathcal{L}_{\boldsymbol{x}}\right\|_2^2 \geq b\gamma^{-1} \sum_{j=1}^{m/b} \left\|b^{-1} \sum_{\boldsymbol{x}\in\mathcal{X}_j} \nabla_{\theta_0(\mathcal{A})}\mathcal{L}_{\boldsymbol{x}}\right\|_2^2 , \tag{6.6}$$

where the last inequality follows from Jensen's inequality. Note that (6.6) provides an approximation of $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ incurring $\mathcal{O}(m/b)$ time because the gradients within a mini-batch can be evaluated in parallel. Moreover, we further approximate the summation over m/b mini-batches in (6.6) by one single uniformly randomly sampled mini-batch $\mathcal{X}_j$. Formally, under the definition of $\|\widetilde{\boldsymbol{\Theta}}_0(\mathcal{A})\|_{\text{tr}} \triangleq \|b^{-1} \sum_{\boldsymbol{x}\in\mathcal{X}_j} \nabla_{\theta_0(\mathcal{A})}\mathcal{L}_{\boldsymbol{x}}\|_2^2$, our final approximation of $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ becomes

$$\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}} \approx m\gamma^{-1}\|\widetilde{\boldsymbol{\Theta}}_0(\mathcal{A})\|_{\text{tr}} . \tag{6.7}$$

This final approximation incurs only $\mathcal{O}(1)$ time and can effectively characterize the performance of neural architectures, as demonstrated in our experiments. Interestingly, a similar form called gradient flow (Wang et al., 2020b) has also been applied in network pruning at initialization.

6.2.3 Optimization and Search Algorithm

The approximation of $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ in Sec. 6.2.2 engenders an efficient optimization of our reformulated NAS in (6.3): Firstly, we apply a penalty method to transform (6.3) into an unconstrained optimization problem. Given a penalty coefficient μ and an exterior penalty function $F(x) \triangleq \max(0, x)$ with a pre-defined constant $\nu \triangleq \gamma n \eta^{-1}$, and a randomly sampled mini-batch $\mathcal{X}_j$, by replacing $\|\boldsymbol{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ with the approximation

in (6.7), our reformulated NAS problem (6.3) can be transformed into

$$\max_{\mathcal{A}} \left[\|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}} - \mu F(\|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}} - \nu) \right] . \tag{6.8}$$

Interestingly, (6.8) implies that the complexity of the final selected architectures is limited by not only the constraint ν (discussed in Sec. 6.2.1) but also the penalty coefficient μ: For a fixed constant ν, a larger μ imposes a stricter limitation on the complexity of architectures (i.e., $\|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}} < \nu$) in the optimization of (6.8).

The optimization of (6.8) in the discrete search space, however, is intractable. So, we apply some optimization tricks to simplify it: Following that of Pham et al. (2018); Liu et al. (2019); Xie et al. (2019b), we represent the search space as an one-shot architecture such that the candidate architectures are subgraphs of this one-shot architecture. Next, instead of optimizing (6.8), we introduce a distribution $p_{\boldsymbol{\alpha}}(\mathcal{A})$ (parameterized by $\boldsymbol{\alpha}$) over the candidate architectures in this search space like that in (Zoph and Le, 2017; Pham et al., 2018; Xie et al., 2019b), and optimize the expected performance of architectures sampled from $p_{\boldsymbol{\alpha}}(\mathcal{A})$:

$$\max_{\boldsymbol{\alpha}} \mathbb{E}_{\mathcal{A}\sim p_{\boldsymbol{\alpha}}(\mathcal{A})} \left[R(\mathcal{A}) \right] \quad \text{s.t.}\, R(\mathcal{A}) \triangleq \|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}} - \mu F(\|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}} - \nu) . \tag{6.9}$$

Then, we apply Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017) to relax the optimization of (6.9) to be continuous and differentiable using the reparameterization trick. Specifically, for a given $\boldsymbol{\alpha}$, to sample an architecture $\mathcal{A}$, we simply have to sample $\boldsymbol{g}$ from $p(\boldsymbol{g}) = \text{Gumbel}(\mathbf{0}, \mathbf{1})$ and then determine $\mathcal{A}$ using $\boldsymbol{\alpha}$ and $\boldsymbol{g}$ (more details in Sec. 6.4.3). Consequently, (6.9) can be transformed into

$$\max_{\boldsymbol{\alpha}} \mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})} \left[R(\mathcal{A}(\boldsymbol{\alpha}, \boldsymbol{g})) \right] . \tag{6.10}$$

After that, we approximate (6.10) based on its first-order Taylor expansion at initialization such that it can be optimized efficiently through a gradient-based algorithm. In particular, given the first-order approximation within the ξ-neighborhood of initialization $\boldsymbol{\alpha}_0$ (i.e., $\|\Delta\|_2 \leq \xi$):

$$\mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}\Big[R(\mathcal{A}(\boldsymbol{\alpha}_0+\Delta,\boldsymbol{g}))\Big] \approx \mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}\Big[R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g})) + \nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}))^\top \Delta\Big], \tag{6.11}$$

the maximum of (6.11) is achieved when

$$\Delta^* = \arg\max_{\|\Delta\|_2\leq\xi} \mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}\Big[\nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}))^\top \Delta\Big] = \xi\, \mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}\left[\frac{\nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}))}{\left\|\mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}[\nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}))]\right\|_2}\right]. \tag{6.12}$$

The closed-form solution in (6.12) follows from the definition of dual norm and requires only a one-step optimization, i.e., without the iterative update of Δ. Similar one-step optimizations have also been adopted by other works (Goodfellow et al., 2015; Wang et al., 2020b).

Unfortunately, the expectation in (6.12) makes the evaluation of Δ^* intractable. Monte Carlo sampling is thus applied to estimate Δ^* efficiently: Given T sequentially sampled $\boldsymbol{g}$ (i.e., $\boldsymbol{g}_1,\ldots,\boldsymbol{g}_T$) and let $G_i \triangleq \nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}_i))$, Δ^* can be approximated as

$$\Delta^* \approx \frac{\xi}{T}\sum_{t=1}^{T} \frac{G_t}{\max(\|G_1\|_2,\ldots,\|G_t\|_2)}. \tag{6.13}$$

Note that the expectation $\left\|\mathbb{E}_{\boldsymbol{g}\sim p(\boldsymbol{g})}[\nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}(\boldsymbol{\alpha}_0,\boldsymbol{g}))]\right\|_2$ in (6.12) has been approximated by $\max(\|G_1\|_2,\ldots,\|G_t\|_2)$ in (6.13) for the sample of $\boldsymbol{g}$ at time t, which is somehow inspired by AMSGrad (Reddi et al., 2018). Interestingly, this approximation is non-decreasing in t and therefore achieves a similar effect of learning rate decay, which may lead to a better-behaved optimization of Δ^*. With the optimal Δ^* and $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}_0 + \Delta^*$, the

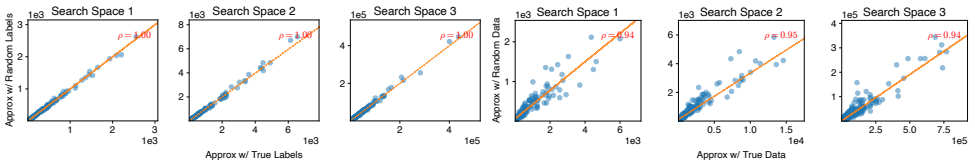
Algorithm 6.1 NAS at Initialization (NASI)

1: **Input:** dataset $\mathcal{D} \triangleq (\mathcal{X}, \mathcal{Y})$, batch size b, steps T, penalty coefficient μ, constraint constant ν, initialized model parameters $\boldsymbol{\theta}_0$ for one-shot architecture and distribution $p_{\boldsymbol{\alpha}_0}(\mathcal{A})$ with initialization $\boldsymbol{\alpha}_0 = \mathbf{0}$, set $\xi = 1$
2: **for** step $t = 1, \ldots, T$ **do**
3: Sample data $\mathcal{D}_t \sim \mathcal{D}$ of size b
4: Sample $\boldsymbol{g}_t \sim p(\boldsymbol{g}) = \text{Gumbel}(\mathbf{0}, \mathbf{1})$ and determine sampled architecture $\mathcal{A}_t$ based on $\boldsymbol{\alpha}_0$, $\boldsymbol{g}_t$
5: Evaluate gradient $G_t = \nabla_{\boldsymbol{\alpha}_0} R(\mathcal{A}_t)$ with data $\mathcal{D}_t$
6: Estimate Δ^* with (6.13) and get $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}_0 + \Delta^*$
7: Select architecture $\mathcal{A}^* = \arg\max_{\mathcal{A}} p_{\boldsymbol{\alpha}^*}(\mathcal{A})$

final architecture can then be selected as $\mathcal{A}^* \triangleq \arg\max_{\mathcal{A}} p_{\boldsymbol{\alpha}^*}(\mathcal{A})$, which completes our *NAS at Initialization* (NASI) algorithm detailed in Algorithm 6.1. Interestingly, this simple and efficient solution in (6.13) can already allow us to select architectures with competitive performances, as shown in our experiments (Sec. 6.5).

6.3 Label- and Data-agnostic Search of NASI

Besides the improved search efficiency by completely avoiding model training during the search, NASI can even guarantee the transferability of its selected architectures with its provable label- and data-agnostic search under mild conditions shown in Sec. 6.3.1 and Sec. 6.3.2, respectively. Particularly, with such provable label- and data-agnostic search, the final selected architectures by NASI on a proxy dataset are also likely to be selected and hence guaranteed to perform well on the target datasets under aforementioned mild conditions. So, the transferability of architectures selected via such label- and data-agnostic search can be guaranteed, as validated in Sec. 6.5 empirically.



(a) Label-agnostic (b) Data-agnostic

Figure 6.1: Comparison of the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ following (6.7) in the three search spaces of NAS-Bench-1Shot1 (Zela et al., 2020b) on CIFAR-10 (a) between random vs. true labels, and (b) between random vs. true data. Each pair (x, y) denotes the approximation of one candidate architecture in the search space with true vs. random labels (or data), respectively. The trends of these approximations are further illustrated by the lines in orange. In addition, Pearson correlation coefficient ρ of the approximations with random vs. true labels (or data) is given in the corner.

6.3.1 Label-Agnostic Search

Our reformulated NAS problem (6.3) explicitly reveals that it can be optimized without the need of the labels from a dataset. Though the approximation of $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ in (6.7) seemingly depends on the labels, (6.7) can, however, be derived using random labels. This is because the Lipschitz continuity assumption on the loss function required by (6.5), which is necessary for the derivation of (6.7), remains satisfied when random labels are used. So, the approximation in (6.7) (and hence our optimization objective (6.8) that is based on this approximation) is label-agnostic, which justifies the label-agnostic nature of NASI. Interestingly, NAS algorithms with a similar label-agnostic search have already been developed in (Liu et al., 2020), which further implies the reasonableness of such label-agnostic search.

The label-agnostic approximation of $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is demonstrated in Fig. 6.1a using the three search spaces of NAS-Bench-1Shot1 with randomly selected labels. According to Fig. 6.1a, the large Pearson correlation coefficient (i.e., $\rho \approx 1$) implies a strong correlation between the approximations with random vs. true labels, which consequently val-

idates the label-agnostic approximation of $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$. Overall, these empirical observations have verified that the approximation of $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ and hence NASI based on the optimization over this approximation are label-agnostic, which will be further validated empirically in Sec. 6.5.4.

6.3.2 Data-Agnostic Search

Besides being label-agnostic, NASI is also guaranteed to be data-agnostic. To justify this, we prove in Proposition 6.2 (following from the notations in Sec. 6.2.1) below that $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is data-agnostic under mild conditions.

Proposition 6.2. *Suppose that $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{x}\|_2 \leq 1$ for all $\boldsymbol{x} \in \mathcal{X}$ given a dataset $(\mathcal{X}, \mathcal{Y})$ of size $|\mathcal{X}| = m$, a given L-layer neural architecture $\mathcal{A}$ is randomly initialized, and the γ-Lipschitz continuous nonlinearity σ satisfies $|\sigma(x)| \leq |x|$. Then, for any two data distributions $P(\boldsymbol{x})$ and $Q(\boldsymbol{x})$, denote $Z \triangleq \int \|P(\boldsymbol{x}) - Q(\boldsymbol{x})\| \, \mathrm{d}\boldsymbol{x}$, as $n_1, \ldots, n_{L-1} \to \infty$ sequentially,*

$$(mn)^{-1} \left| \left\| \mathbf{\Theta}_0(\mathcal{A}; P) \right\|_{\text{tr}} - \left\| \mathbf{\Theta}_0(\mathcal{A}; Q) \right\|_{\text{tr}} \right| \leq n_0^{-1} Z D(\gamma)$$

with probability arbitrarily close to 1. $D(\gamma)$ is set to L if $\gamma = 1$, and $(1 - \gamma^{2L})/(1 - \gamma^2)$ otherwise.

Its proof is in Appendix C.1.4. Proposition 6.2 reveals that for any neural architecture $\mathcal{A}$, $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is data-agnostic if either one of the following conditions is satisfied: (a) Different datasets achieve a small Z or (b) the input dimension n_0 is large. Interestingly, these two conditions required by the data-agnostic $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ can be well-satisfied in practice. Firstly, we always have $Z < 2$ according to the property of probability distributions. Moreover, many real-world datasets are of high dimensions such as $\sim 10^3$ for CIFAR-10 (Krizhevsky et al., 2009) and $\sim 10^5$ for COCO

(Lin et al., 2014). Since $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ under such mild conditions is data-agnostic, NASI using $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ as the optimization objective in (6.3) is also data-agnostic.

While $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is costly to evaluate, we demonstrate in Fig. 6.1b that the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ in (6.7) is also data-agnostic using random data following the standard Gaussian distribution. Similar to the results using true vs. random labels in Fig. 6.1a, the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ using random vs. true data are also highly correlated with a large Pearson correlation coefficient (i.e., $\rho > 0.9$). Interestingly, the correlation here is slightly smaller than the label-agnostic approximations in Fig. 6.1a, which implies that the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is more agnostic to the labels than data. Based on these results, the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ is guaranteed to be data-agnostic. So, NASI based on the optimization over such a data-agnostic approximation is also data-agnostic, which will be further validated empirically in Sec. 6.5.4.

6.4 Experimental Settings

6.4.1 Determination of Constraint ν and Penalty Coefficient μ

As demonstrated in Sec. 6.2.1, constraint ν (derived from $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}} < mn\eta^{-1}$ in (6.3)) introduces a trade-off between the complexity of final selected architectures and the optimization behavior in their model training. This constraint ν hence is of great importance to our NASI algorithm. Though ν has already been pre-defined as $\nu \triangleq \gamma n\eta^{-1}$ in Sec. 6.2.3, we still tend to take it as a hyper-parameter to be determined for the selection of best-performing architectures in practice. Specifically, in this work,

two methods are adopted to determine ν during the search progress: The **fixed** and **adaptive** method shown as below. Notably, the final architectures selected by NASI via the **fixed** and **adaptive** method are called NASI-FIX and NASI-ADA, respectively. Note that our experiments in the main text suggest that both methods can select architectures with competitive generalization performance over different tasks.

The fixed determination. We initialize and fix ν with ν_0 during the whole search process. Hence, ν_0 is required to provide a good trade-off between the complexity of architectures and their optimization behavior in the search space. Intuitively, the expectation of architecture complexity in the search space can help to select architectures with medium complexity and hence implicitly achieve a good trade-off between the complexity of architectures and their optimization behavior. Specifically, we randomly sample $N = 50$ architectures in the search space (i.e., $\mathcal{A}_1, \cdots, \mathcal{A}_N$), and then determine ν_0 before the search process by

$$\nu = \nu_0 = N^{-1} \sum_{i}^{N} \|\widetilde{\mathbf{\Theta}}_0(\mathcal{A}_i)\|_{\text{tr}} \,. \tag{6.14}$$

Note that we can further enlarge ν_0 in practice to encourage the selection of architectures with larger complexity.

The adaptive determination. We initialize ν with a relatively large ν_0 and then adaptively update it with the expected $\|\widetilde{\mathbf{\Theta}}_0(\mathcal{A})\|_{\text{tr}}$ of sampled architectures during the search process. Specifically, with sampled architectures $\mathcal{A}_1, \cdots, \mathcal{A}_t$ in the history, ν at time t (i.e., ν_t) during the search

process is given by

$$\nu_t = t^{-1}\left(\nu_0 + \sum_{\tau=1}^{t-1} \|\widetilde{\Theta}_0(\mathcal{A}_\tau)\|_{\text{tr}}\right). \tag{6.15}$$

We apply a relatively large ν_0 to ensure a loose constraint on the complexity of architectures in the first several steps of the search process. Note that the adaptive determination provides a more accurate approximation of the expected complexity of architectures in the search space than the fixed determination method if more architectures are sampled to update ν_t, i.e., $t > N$.

Note that (6.8) in Sec. 6.2.3 further reveals the limitation on the complexity of final selected architectures by the penalty coefficient μ. Particularly, μ=0 indicates no limitation on the complexity of architectures. Following from the introduced trade-off between the complexity of final selected architectures and the optimization behavior in their model training by the constraint ν, for a search space with relatively larger-complexity architectures, a larger penalty coefficient μ (i.e., $\mu = 2$) is preferred to search for architectures with relatively smaller complexity to ensure a well-behaved optimization with a larger learning rate η. On the contrary, for a search space with relatively smaller-complexity architectures, a lower penalty coefficient μ (i.e., μ=1) is adopted to ensure the complexity and hence the representation power of the final selected architectures. Sec. 6.6.4 provides further ablation study on the constraint ν and the penalty coefficient μ.

6.4.2 The DARTS Search Space

Following from the DARTS (Liu et al., 2019) search space, candidate architecture in our search space comprise a stack of L cells, and each

cell can be represented as a directed acyclic graph (DAG) of N nodes denoted by $\{x_0, x_1, \ldots, x_{N-1}\}$. To select the best-performing architectures, we instead need to select their corresponding cells, including the normal and reduction cell. Specifically, x_0 and x_1 denote the input nodes, which are the output of two preceding cells, and the output x_N of a cell is the concatenation of all intermediate nodes from x_2 to x_{N-1}. Following that of SNAS (Xie et al., 2019b), by introducing the distribution of architecture in the search space (i.e., $p_{\alpha}(\mathcal{A})$ in (6.9)), each intermediate nodes x_i ($2 \leq i \leq N-1$) denotes the output of a single sampled operation $o_i \sim p_i(o)$ with $\sum_{o \in \mathcal{O}} p_i(o) = 1$ given a single sampled input $x_j \sim p_i(x)$ with $j \in 0, \cdots, i-1$ and $\sum_{j=0}^{i-1} p_i(x_j) = 1$, where $\mathcal{O}$ is a predefined operation set. After the search process, only operations achieving the top-2 largest $p_i(o)$ and inputs achieving the top-2 largest $p_i(x)$ for node x_i are retained. Each intermediate node x_j hence connects to two preceding nodes with the corresponding selected operations.

Following from DARTS, the candidate operation set $\mathcal{O}$ includes following operations: 3×3 max pooling, 3×3 avg pooling, identity, 3×3 separable conv, 5×5 separable conv, 3×3 dilated separable conv, 5×5 dilated separable conv. Note that our search space has been modified slightly based on the standard DARTS (Liu et al., 2019) search space: (a) Operation *zero* is removed from the candidate operation set in our search space since it can never been selected in the standard DARTS search space, and (b) the inputs of each intermediate node are selected independently from the selection of operations, while DARTS attempts to select the inputs of intermediate nodes by selecting their coupling operations with the largest weights. Notably, following from DARTS, we need to search for two different cells: A normal cell and a reduction cell. Besides, a max-pooling operation in between normal and reduction

cell is applied to down-sampling intermediate features during the search process inspired by NAS-Bench-1Shot1 (Zela et al., 2020b).

6.4.3 Sampling Architecture with Gumbel-Softmax

Notably, aforementioned $p_i(o)$ and $p_i(x)$ for the node x_i follow the categorical distributions. To relax the optimization of (6.9) with such categorical distributions to be continuous and differentiable, Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017) is applied. Specifically, supposing $p_i(o)$ and $p_i(x)$ are parameterized by $\alpha^o_{i,j}$ and $\alpha^x_{i,k}$ respectively, with Straight-Through (ST) Gumbel Estimator (Jang et al., 2017), we can sample the single operation and input for the node x_i $(2 \leq i \leq N-1)$ during the search process by

$$\begin{aligned} j^* &= \arg\max_j \frac{\exp((\alpha^o_{i,j} + g^o_{i,j})/\tau)}{\sum_j \exp((\alpha^o_{i,j} + g^o_{i,j})/\tau)} \\ k^* &= \arg\max_k \frac{\exp((\alpha^x_{i,k} + g^x_{i,k})/\tau)}{\sum_k \exp((\alpha^x_{i,k} + g^x_{i,k})/\tau)}, \end{aligned} \tag{6.16}$$

where $g^o_{i,j}$ and $g^x_{i,k}$ are sampled from Gumbel(0, 1) and τ denotes the softmax temperature, which is conventionally set to be 1 in our experiments. Note that in (6.16), $j \in [0, \cdots, |\mathcal{O}|-1]$ and $k \in [0, \cdots, i-1]$, which correspond to the $|\mathcal{O}|$ operations in the operation set $\mathcal{O}$ and the i candidate inputs for the node x_i, respectively. Notably, the gradient through ST can be approximated by its continuous counterpart as suggested in (Jang et al., 2017), thus allowing the continuous and differentiable optimization of (6.9). By sampling the discrete operation and input with (6.16) for each nodes in the cells, the final sampled architecture can hence be determined.

6.4.4 Evaluation on CIFAR-10/100 and ImageNet

Evaluation on CIFAR-10/100. Following DARTS (Liu et al., 2019), the final selected architectures consist of 20 searched cells: 18 of them are identical normal cell and 2 of them are identical reduction cell. An auxiliary tower with weight 0.4 is located at 13-th cell of the final selected architectures and the number of initial channels is set to be 36. The final selected architecture are then trained via stochastic gradient descent (SGD) of 600 epochs with a learning rate cosine scheduled from 0.025 to 0, momentum 0.9, weight decay 3×10^{-4} and batch size 96 on a single Nvidia 2080Ti GPU. Cutout (Devries and Taylor, 2017), and ScheduledDropPath linearly increased from 0 to 0.2 are also employed for regularization purpose.

Evaluation on ImageNet. We evaluate the transferability of the selected architectures from CIFAR-10 to ImageNet. The architecture comprises of 14 cells (12 normal cells and 2 reduction cells). To evaluate in the mobile setting (under 600M multiply-add operations), the number of initial channels is set to 46. We adopt conventional training enhancements (Liu et al., 2019; Chen et al., 2019; Chen and Hsieh, 2020) include an auxiliary tower loss of weight 0.4 and label smoothing. Following P-DARTS (Chen et al., 2019) and SDARTS-ADV (Chen and Hsieh, 2020), we train the model from scratch for 250 epochs with a batch size of 1024 on 8 Nvidia 2080Ti GPUs. The learning rate is warmed up to 0.5 for the first 5 epoch and then decreased to zero linearly. We adopt the SGD optimizer with 0.9 momentum and a weight decay of 3×10^{-5}.

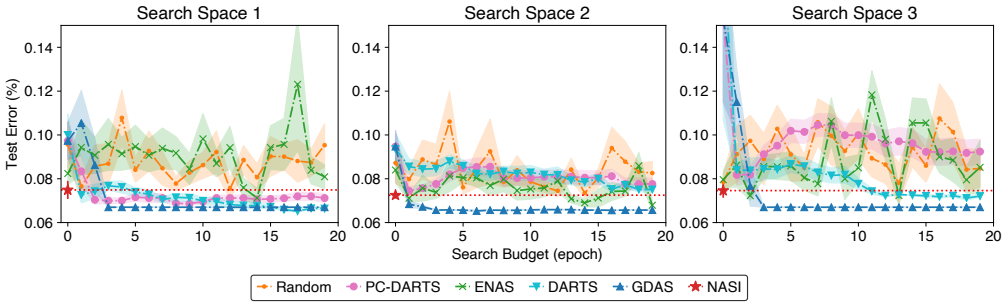


Figure 6.2: Comparison of search efficiency (search budget in x-axis) and effectiveness (test error evaluated on CIFAR-10 in y-axis) between NASI and other NAS algorithms in the three search spaces of NAS-Bench-1Shot1. The test error for each algorithm is reported with the mean and standard error after ten independent searches.

6.5 Experiments

6.5.1 Search in NAS-Bench-1Shot1

We firstly validate the search efficiency and effectiveness of NASI in the three search spaces of NAS-Bench-1Shot1 (Zela et al., 2020b) on CIFAR-10. As the three search spaces are relatively small, a lower penalty coefficient μ and a larger constraint ν (i.e., μ=1 and ν=1000) are adopted to encourage the selection of high-complexity architectures in the optimization of (6.8). Here, ν is determined adaptively as shown in Sec. 6.4.1.

Figure 6.2 shows the results comparing the efficiency and effectiveness between NASI with a one-epoch search budget and other NAS algorithms with a maximum search budget of 20 epochs to allow sufficient model training during their search process. Figure 6.2 reveals that among all these three search spaces, NASI consistently selects architectures of better generalization performance than other NAS algorithms with a search budget of only one epoch. Interestingly, the selected architectures by the one-epoch NASI achieve performances that are comparable to the best-performing NAS algorithms with 19× more search budget. Above all, NASI guarantees its benefits of improving the search efficiency of NAS al-

gorithms considerably without sacrificing the generalization performance of its selected architectures.

6.5.2 Search in NAS-Bench-201

To further justify the improved search efficiency and competitive search effectiveness of our NASI algorithm, we also compare it with other state-of-the-art NAS algorithms in NAS-Bench-201 (Dong and Yang, 2020) on CIFAR-10/100 (C10/100) and ImageNet-16-200 (IN-16-200) [4]. Table 6.1 summarizes the comparison. Note that the baselines in Table 6.1 are obtained from TE-NAS (Chen et al., 2021) paper. Notably, compared with training-based NAS algorithms, our NASI algorithm can achieve significantly improved search efficiency while maintaining a competitive or even outperforming test performance. Furthermore, our NASI algorithm is shown to be able to enjoy both improved search efficiency and effectiveness when compared with most other training-free baselines. Although TE-NAS, as the best-performing training-free NAS algorithm on both CIFAR-10/100, achieves a relatively improved test accuracy than our NASI (T), our NASI with a search budget of $T = 30s$ is 50× more efficient than TE-NAS and is able to achieve compelling test performance on all the three datasets. Moreover, by providing a larger search budget, our NASI algorithm (i.e., NASI ($4T$)) can in fact achieve comparable (on CIFAR-10/100) or even better (on ImageNet-16-120) search results with 12× lower search cost compared with TE-NAS.

[4] ImageNet-16-200 is a down-sampled variant of ImageNet (ImageNet16×16) (Chrabaszcz et al., 2017)

Table 6.1: The comparison among state-of-the-art (SOTA) NAS algorithms on NAS-Bench-201. The performance of the final architectures selected by NASI is reported with the mean and standard deviation of four independent trials. The search costs are evaluated on a single Nvidia 1080Ti.

Architecture	Test Accuracy (%)			Search Cost (GPU Sec.)	Search Method
	C10	C100	IN-16-120		
ResNet (He et al., 2016)	93.97	70.86	43.63	-	-
ENAS (Pham et al., 2018)	54.30	15.61	16.32	13315	RL
DARTS (1st) (Liu et al., 2019)	54.30	15.61	16.32	10890	gradient
DARTS (2nd) (Liu et al., 2019)	54.30	15.61	16.32	29902	gradient
GDAS (Dong and Yang, 2019b)	93.61±0.09	70.70±0.30	41.84±0.90	28926	gradient
NASWOT (N=10) (Mellor et al., 2020a)	92.44±1.13	68.62±2.04	41.31±4.11	3	training-free
NASWOT (N=100) (Mellor et al., 2020a)	92.81±0.99	69.48±1.70	43.10±3.16	30	training-free
NASWOT (N=1000) (Mellor et al., 2020a)	92.96±0.81	69.98±1.22	44.44±2.10	306	training-free
TE-NAS (Chen et al., 2021)	93.90±0.47	71.24±0.56	42.38±0.46	1558	training-free
KNAS (Xu et al., 2021)	93.05	68.91	34.11	4200	training-free
NASI (T)	93.08±0.24	69.51±0.59	40.87±0.85	30	training-free
NASI ($4T$)	93.55±0.10	71.20±0.14	44.84±1.41	120	training-free

6.5.3 Search in the DARTS Search Space

We then compare NASI with other NAS algorithms in a more complex search space than NAS-Bench-1Shot1, i.e., the DARTS (Liu et al., 2019) search space (detailed in Sec. 6.4.2). Here, NASI selects the architecture with a search budget of T=100, batch size of b=64 and μ=2. Besides, two different methods are applied to determine the constraint ν during the search process: the adaptive determination with an initial value of 500 and the fixed determination with a value of 100. The final selected architectures with adaptive and fixed ν are, respectively, called NASI-ADA and NASI-FIX (visualized in Sec. 6.6.3), which are then evaluated on CIFAR-10/100 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) following Sec. 6.4.4.

Table 6.2 summarizes the generalization performance of the final architectures selected by various NAS algorithms on CIFAR-10/100. Compared with popular training-based NAS algorithms, NASI achieves a substantial improvement in search efficiency and maintains a competitive generalization performance. Even when compared with the training-free

Table 6.2: Performance comparison among state-of-the-art (SOTA) image classifiers on CIFAR-10/100. The performance of the final architectures selected by NASI is reported with the mean and standard deviation of five independent evaluations. The search costs are evaluated on a single Nvidia 1080Ti.

Architecture	Test Error (%)		Params (M)		Search Cost (GPU Hours)	Search Method
	C10	C100	C10	C100		
DenseNet-BC (Huang et al., 2017b)	3.46*	17.18*	25.6	25.6	-	manual
NASNet-A (Zoph et al., 2018)	2.65	-	3.3	-	48000	RL
AmoebaNet-A (Real et al., 2019b)	3.34±0.06	18.93†	3.2	3.1	75600	evolution
PNAS (Liu et al., 2018)	3.41±0.09	19.53*	3.2	3.2	5400	SMBO
ENAS (Pham et al., 2018)	2.89	19.43*	4.6	4.6	12	RL
NAONet (Luo et al., 2018a)	3.53	-	3.1	-	9.6	NAO
DARTS (2nd) (Liu et al., 2019)	2.76±0.09	17.54†	3.3	3.4	24	gradient
GDAS (Dong and Yang, 2019b)	2.93	18.38	3.4	3.4	7.2	gradient
NASP (Yao et al., 2020)	2.83±0.09	-	3.3	-	2.4	gradient
P-DARTS (Chen et al., 2019)	2.50	-	3.4	-	7.2	gradient
DARTS- (avg) (Chu et al., 2020)	2.59±0.08	17.51±0.25	3.5	3.3	9.6	gradient
SDARTS-ADV (Chen and Hsieh, 2020)	2.61±0.02	-	3.3	-	31.2	gradient
R-DARTS (L2) (Zela et al., 2020a)	2.95±0.21	18.01±0.26	-	-	38.4	gradient
TE-NAS♯ (Chen et al., 2021)	2.83±0.06	17.42±0.56	3.8	3.9	1.2	training-free
NASI-FIX	2.79±0.07	16.12±0.38	3.9	4.0	**0.24**	training-free
NASI-ADA	2.90±0.13	16.84±0.40	3.7	3.8	**0.24**	training-free

† Reported by Dong and Yang (2019b) with their experimental settings.
* Obtained by training corresponding architectures without cutout (Devries and Taylor, 2017) augmentation.
♯ Evaluated using our experimental settings in Sec. 6.4.4.

NAS algorithm (i.e., TE-NAS), NASI is also able to select competitive or even outperformed architectures with a smaller search cost. Besides, NASI-FIX achieves the smallest test error on CIFAR-100, which demonstrates the transferability of the architectures selected by NASI over different datasets. We also evaluate the performance of the final architectures selected by NASI on ImageNet and summarize the results in Table 6.3. Notably, NASI-FIX and NASI-ADA outperform the expert-designed architecture ShuffleNet 2×(v2), NAS-based architecture MnasNet-92 and DARTS by a large margin, and are even competitive with best-performing one-shot NAS algorithm DARTS-. Notably, while achieving better generalization performance than TE-NAS (ImageNet), NASI (C10) is even shown to be more efficient by directly transferring the architectures selected on CIFAR-10 to ImageNet based on its provable transferability in Sec. 6.3. Meanwhile, by directly searching on ImageNet, NASI-ADA (ImageNet) is able to achieve further improved performance over NASI-ADA (C10)

Table 6.3: Performance comparison among SOTA image classifiers on ImageNet. Note that architectures followed by C10 are transferred from the CIFAR-10 dataset, while architectures followed by ImageNet are directly selected on ImageNet.

Architecture	Test Error (%)		Params (M)	+× (M)	Search Cost (GPU Days)
	Top-1	Top-5			
Inception-v1 (Szegedy et al., 2015a)	30.1	10.1	6.6	1448	-
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	-
ShuffleNet 2×(v2) (Ma et al., 2018)	25.1	7.6	7.4	591	-
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000
AmoebaNet-A (Real et al., 2019b)	25.5	8.0	5.1	555	3150
PNAS (Liu et al., 2018)	25.8	8.1	5.1	588	225
MnasNet-92 (Tan et al., 2019b)	25.2	8.0	4.4	388	-
DARTS (Liu et al., 2019)	26.7	8.7	4.7	574	4.0
SNAS (mild) (Xie et al., 2019b)	27.3	9.2	4.3	522	1.5
GDAS (Dong and Yang, 2019b)	26.0	8.5	5.3	581	0.21
ProxylessNAS (Cai et al., 2019b)	24.9	7.5	7.1	465	8.3
P-DARTS (Chen et al., 2019)	24.4	7.4	4.9	557	0.3
DARTS- (Chu et al., 2020)	23.8	7.0	4.5	467	4.5
SDARTS-ADV (Chen and Hsieh, 2020)	25.2	7.8	5.4	594	1.3
TE-NAS (C10) (Chen et al., 2021)	26.2	8.3	5.0	-	0.05
TE-NAS (ImageNet) (Chen et al., 2021)	24.5	7.5	5.4	-	0.17
NASI-FIX (C10)	24.3	7.3	5.2	585	**0.01**
NASI-FIX (ImageNet)	24.4	7.4	5.5	615	**0.01**
NASI-ADA (C10)	25.0	7.8	4.9	559	**0.01**
NASI-ADA (ImageNet)	24.8	7.5	5.2	585	**0.01**

while the performance of NASI-FIX (ImageNet) and NASI-FIX (C10) are quite similar.[5] Above all, the results on ImageNet further confirm the good transferability of the architectures selected by NASI to larger-scale datasets.

6.5.4 Label- and Data-Agnostic Search

To further validate the label- and data-agnostic search achieved by our NASI as discussed in Sec. 6.3, we compare the generalization performance of the final architectures selected by NASI using random labels and data on CIFAR-10. The random labels are randomly selected from all possible

[5] In order to maintain the same initial channels between NASI-FIX (ImageNet) and NASI-FIX (C10), the multiply-add operations of NASI-FIX (ImageNet) has to be larger than 600M by a small margin.

Table 6.4: Performance comparison of architectures selected with random or true labels/data by NASI on CIFAR-10. The standard method denotes the search with the true labels and data of CIFAR-10 and each test error is reported with the mean and standard deviation of five independent searches.

Method	NAS-Bench-1Shot1			DARTS
	S1	S2	S3	
Standard	7.3±1.1	7.2±0.4	7.2±0.6	2.95±0.13
Random Label	6.8±0.3	7.0±0.4	7.5±1.4	2.90±0.12
Random Data	6.6±0.2	7.5±0.7	7.3±0.9	2.97±0.10

categories while the random data is i.i.d. sampled from the standard Gaussian distribution. Both NAS-Bench-1Shot1 and the DARTS search space are applied in this performance comparison where the same search and training settings in Sec. 6.5.1 and Sec. 6.5.3 are adopted.

Table 6.4 summarizes the performance comparison. Interestingly, among all the four search spaces, comparable generalization performances are obtained on CIFAR-10 for both the architectures selected with random labels (or data) and the ones selected with true labels and data. These results hence confirm the label- and data-agnostic search achieved by NASI, which therefore also further validates the transferability of the architectures selected by NASI over different datasets.

6.6 More Results

6.6.1 Trade-off Between Model Complexity and Optimization Behaviour

In this section, we empirically validate the existence of trade-off between model complexity of selected architectures and the optimization behavior in their model training for our reformulated NAS detailed in Sec. 6.2.1.

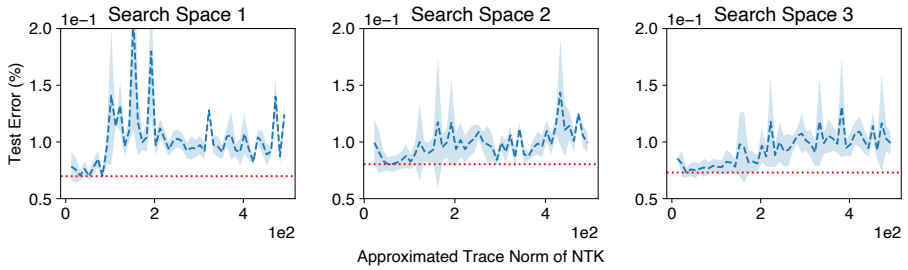


Figure 6.3: The relation between test error and approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ of candidate architectures in the three search spaces of NAS-Bench-1Shot1 over CIFAR-10. Note that x-axis denotes the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$, which is averaged over the architectures grouped in the same bin based on their approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$. Correspondingly, y-axis denotes the averaged test error with standard deviation (scaled by 0.5) of these grouped architectures. In addition, the red lines demonstrate the smallest test errors achieved by candidate architectures in these three search spaces.

The trade-off in NAS-Bench-1Shot1. Specifically, Figure 6.3 illustrates the relation between test error and approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ of candidate architectures in the three search spaces of NAS-Bench-1Shot1. Note that with the increasing of the approximated $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$, the test error decreases rapidly to a minimum and then increase gradually, which implies that architectures only with certain $\|\mathbf{\Theta}_0(\mathcal{A})\|_{\text{tr}}$ (or with desirable complexity instead of the largest complexity) can achieve the best generalization performance. These results hence validate the existence of trade-off between the model complexity and the generalization performance of selected architectures. Note that such trade-off usually results from different optimization behavior in the model training of those selected architectures as demonstrated in the following experiments.

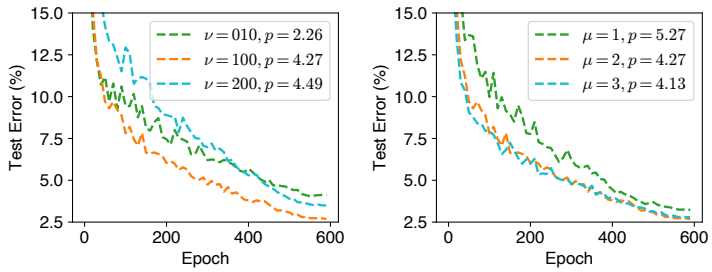
The trade-off in the DARTS search space. We then illustrate the optimization behavior of the final selected architecture from the DARTS search space with different constraint and penalty coefficient in Figure 6.4 to further confirm the existence of such trade-off in our reformulated NAS (6.3). Specifically, we apply NASI with the fixed determination of

ν to select architectures in the the DARTS search space over CIFAR-10, where ν_0 in the fixed determination method introduced in Sec. 6.4.1 is modified manually for the comparison. Besides, a search budget of $T = 100$ and batch size of $b = 64$ are adopted. These final selected architectures are then trained on CIFAR-10 following Sec. 6.5.3.

Note that, in Figure 6.4(a), with the increasing of ν, the final selected architectures by NASI contain more parameters and hence achieve larger complexity. Meanwhile, the final selected architecture with ν=10 enjoys a faster convergence rate in the first 50 epochs, but a poorer generalization performance than the one with ν= 200. Interestingly, the final selected architecture with ν=100 realizes the fastest convergence and the best generalization performance by achieving a proper complexity of final selected architectures. These results validate the existence and also the importance of the trade-off between the complexity of final selected architectures and the optimization behavior in their model training. However, the trade-off introduced by the penalty coefficient μ shown in Figure 6.4(b) is hard to be observed, which indicates that the constraint ν is of greater importance than the penalty coefficient μ in terms of the trade-off between the complexity of architectures and their optimization behavior. Interestingly, Figure 6.4(b) can still reveal the slower convergence rate and poorer generalization performance caused by the selection of architecture with relatively larger complexity, i.e., μ=1.

6.6.2 Comparison to Other Training-Free Metrics

We compare our methods with other training-free NAS methods using both the Spearman correlation and the Kendall's tau between the training-free metrics and the test accuracy on CIFAR-10 in the three search spaces of NAS-Bench-1Shot1. We adopt one uniformly randomly sampled mini-



(a) Constraint ν (b) Penalty coefficient μ

Figure 6.4: The optimization behavior (test error on CIFAR-10 in model training) of the final selected architectures with different constraint ν and penalty coefficient μ. The model parameter (MB) denoted by p of each architecture is given in the top-right corner.

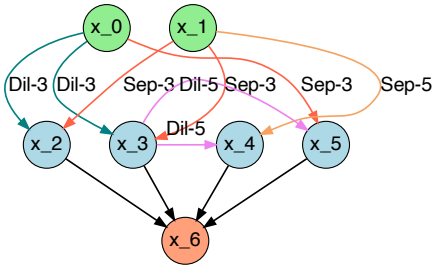
Table 6.5: Comparison of the Spearman correlation and the Kendall's tau for various training-free metrics in the three search spaces of NAS-Bench-1Shot1 on CIFAR-10.

Methods	**Spearman Correlation**			**Kendall's Tau**		
	S1	**S2**	**S3**	**S1**	**S2**	**S3**
SNIP (Lee et al., 2019b)	−0.49	−0.62	−0.79	−0.39	−0.49	−0.63
GraSP (Wang et al., 2020b)	0.41	0.54	0.17	0.33	0.42	0.15
SynFlow (Tanaka et al., 2020)	−0.52	−0.45	−0.53	−0.42	−0.40	−0.47
NASWOT (Mellor et al., 2020a)	0.21	0.32	0.54	0.16	0.24	0.44
NASI (conditioned)	**0.62**	**0.74**	**0.76**	**0.44**	**0.53**	**0.53**

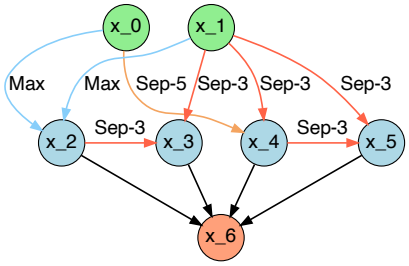
batch data to evaluate these two correlations and apply the same implementations of these training-free NAS methods in (Abdelfattah et al., 2021) for the comparison. Table 6.5 summarizes the comparison, where the results of our metric are reported under the constraint in (6.3). Interestingly, our metric generally achieves a higher positive correlation than other training-free metrics, which confirms the reasonableness and also the effectiveness of our training-free metric.

6.6.3 Architectures Selected by NASI

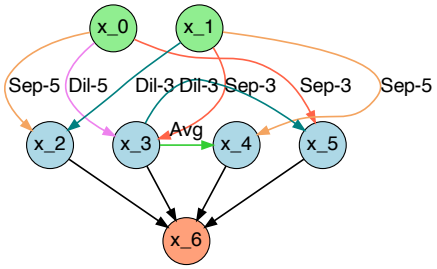
The final selected cells in the DARTS search space (i.e., NASI-FIX and NASI-ADA used in Sec. 6.5.3) are illustrated in Figure 6.5, where different operations are denoted with different colors for clarification. Interest-



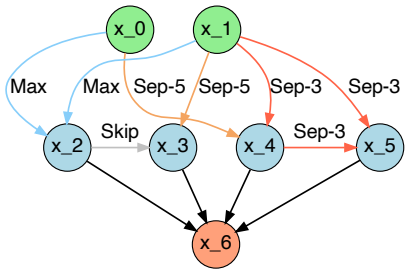
(a) NASI-FIX (Normal)



(b) NASI-FIX (Reduction)



(c) NASI-ADA (Normal)



(d) NASI-ADA (Reduction)

Figure 6.5: The final selected normal and reduction cells of NASI-FIX and NASI-ADA in the reduced DARTS search space on CIFAR-10. Note that x_0, x_1 denote the input nodes, x_2, x_3, x_4, x_5 denote intermediate nodes and x_6 denotes the output node as introduced in Sec. 6.4.2.

ingly, according to the definitions and findings in (Shu et al., 2020), these final selected cells by NASI are relatively deeper and shallower than the ones selected by other NAS algorithms and hence may achieve worse generalization performance. Nonetheless, in our experiments, the architectures constructed with these cells (i.e., NASI-FIX and NASI-ADA) achieve competitive or even better generalization performance than the ones selected by other NAS algorithms as shown in Table 6.2 and Table 6.3. A possible explanation is that our NASI algorithm provides a good trade-off between the complexity of architectures and the optimization in their model training, while other NAS algorithms implicitly prefer architectures with smaller complexity and faster convergence rate as revealed in (Shu et al., 2020). Note that the final selected cells in NASI-FIX and NASI-ADA are of great similarity to each other, which hence implies that the adaptive determination and the fixed determination of constraint ν share similar effects on the selection of final architectures.

6.6.4 Ablation Studies and Discussions

The impacts of architecture width. As our theoretically grounded performance estimation of neural architectures relies on an infinite width assumption (i.e., $n \to \infty$ in Proposition 6.1), we investigate the impacts of varying architecture width on the generalization performance of final selected architectures by our NASI. We adopt the same search settings in Sec. 6.5.1 but with a varying architecture width (i.e., a varying number of initial channels) on CIFAR-10 for the three search spaces of NAS-Bench-1Shot1. Table 6.6 shows results of the effectiveness of our NASI in NAS-Bench-1Shot1 with varying architecture widths N: NASI consistently selects well-performing architectures and a larger width ($N = 32$) enjoys better search results. Hence, the infinite width assumption for NTK does

Table 6.6: Search with varying architecture widths N in the three search spaces of NAS-Bench-1Shot1.

Spaces	$N = 2$	$N = 4$	$N = 8$	$N = 16$	$N = 32$
S1	7.0±0.6	8.3±1.8	8.0±2.2	7.3±1.5	6.5±0.2
S2	7.3±0.7	8.0±1.5	7.3±0.6	7.3±0.4	7.0±0.2
S3	7.6±0.8	7.7±1.1	6.8±0.1	6.8±0.3	6.4±0.2

not cause any empirical issues for our NASI.

The effectiveness of NTK trace norm approximations. Since the trace norm of NTK is costly to evaluate, we have provided our approximation to it in Sec. 6.2.2. In this section, we empirically validate the effectiveness of our NTK trace norm approximation. Specifically, we evaluate the Pearson correlation between our approximations (including the approximations using the sum of sample gradient norm in the first inequality of (6.6) and the approximations using mini-batch gradient norm in (6.7)) and the exact NTK trace norm under varying batch size in the three search spaces of NAS-Bench-1Shot1. The results are summarized in Table 6.7. The results confirm that our approximation is reasonably good to estimate the exact NTK trace norm of different architectures by achieving a high Pearson correlation between our approximations and the exact NTK trace norm. Interestingly, a larger batch size of mini-batch gradient norm generally achieves a better approximation, and the sum of sample gradient norm achieves the best approximation, which can be explained by the possible approximation errors we introduced when deriving our (6.6) and (6.7).

The impacts of batch size. Since we only adopt a mini-batch to approximate the NTK trace norm shown in Sec. 6.2.2, we further examine the impacts of varying batch size on the search results in the three search

Table 6.7: Pearson correlation between our NTK trace norm approximation and the exact NTK trace norm in the three search spaces of NAS-Bench-1Shot1.

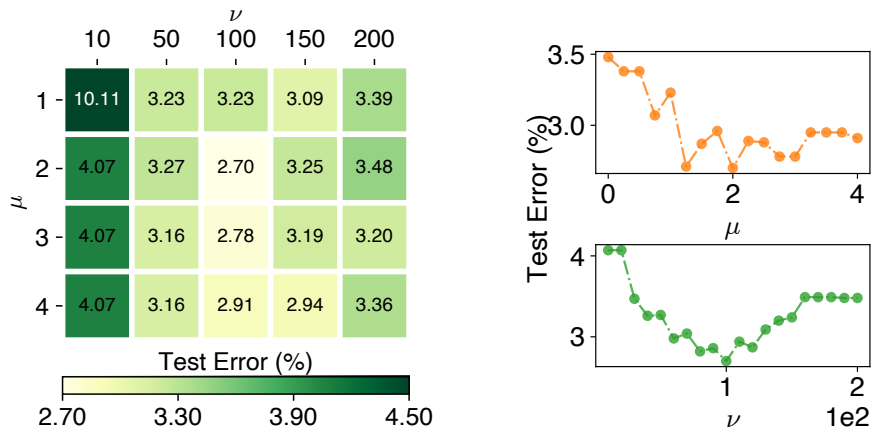
Spaces	**Mini-batch**						**Sum**
	$b = 4$	$b = 8$	$b = 16$	$b = 32$	$b = 64$	$b = 128$	
S1	0.74	0.80	0.84	0.83	0.85	0.86	0.88
S2	0.82	0.87	0.90	0.89	0.91	0.91	0.92
S3	0.66	0.74	0.81	0.79	0.82	0.84	0.87

Table 6.8: Search with varying batch sizes b in the three search spaces of NAS-Bench-1Shot1.

Spaces	$b = 8$	$b = 16$	$b = 32$	$b = 64$	$b = 128$
S1	6.8±0.3	6.7±0.2	6.7±0.1	6.0±0.3	7.0±0.3
S2	6.9±0.2	7.3±0.4	7.1±0.2	7.2±0.3	6.8±0.2
S3	7.0±0.3	6.6±0.2	6.7±0.2	6.6±0.1	7.1±0.2

spaces of NAS-Bench-1Shot1 in this section. Table 6.8 summarizes the search results. Interestingly, the results show that our approximation under varying batch sizes achieves comparable search results, further confirming the effectiveness of our approximations in select well-performing architectures.

The impacts of constraint ν and penalty coefficient μ. Based on the analysis in Sec. 6.2.3 and the results in Sec. 6.6.1, the choice of constraint ν and penalty coefficient μ is thus non-trivial to select best-performing architectures since they trade off the complexity of final selected architectures and the optimization in their model training. In this section, we demonstrate their impacts on the generalization performance of the finally selected architectures and also the effectiveness of our fixed determination on the constraint ν in detail. Notably, we adopt the same settings (including the search and training settings) as those in Sec. 6.6.1 on the DARTS search space, where ν_0 in the fixed determination method



(a) Joint impacts (b) Individual impacts

Figure 6.6: The impacts of constraint ν and penalty coefficient μ on the generalization performance of the final selected architectures by NASI: (a) their joint impacts, and (b) their individual impacts.

introduced in Sec. 6.4.1 is modified manually for the comparison.

Figure 6.6 summarizes their impacts into two parts: the joint impacts in Figure 6.6(a) and the individual impacts in Figure 6.6(b). Notably, in both plots, with the increasing of ν or μ, the test error of the final selected architecture by NASI gradually decreases to a minimal and then increase steadily, hence further validating the existence of the trade-off introduced by ν and μ. Moreover, the final selected architectures with ν=100 significantly outperform other selected architectures. This result thus supports the effectiveness of our fixed determination method for ν in NASI. Interestingly, Figure 6.6(b) reveals a less obvious decreasing trend of test error for μ compared with ν, which hence further implies the greater importance of constraint ν than penalty coefficient μ in terms of the generalization performance of the final selected architectures. Therefore, in our experiments, we conventionally set penalty coefficient μ=1 for small-complexity search spaces and μ=2 for large-complexity search spaces as introduced in Sec. 6.4.1.

6.7 Conclusion

This work describes a novel NAS algorithm called NASI that exploits the capability of NTK for estimating the performance of candidate architectures at initialization. Consequently, NASI can completely avoid model training during the search process to achieve higher search efficiency than existing NAS algorithms. NASI can also achieve competitive generalization performance across different search spaces and benchmark datasets. Interestingly, NASI is guaranteed to be label- and data-agnostic under mild conditions, which implies the transferability of the final architectures selected by NASI over different datasets. With all these advantages, NASI can thus be adopted to select well-performing architectures for unsupervised tasks and larger-scale datasets efficiently, which to date remains challenging to other training-based NAS algorithms. Furthermore, NASI can also be integrated into other training-based one-shot NAS algorithms to improve their search efficiency while preserving the search effectiveness of these training-based algorithms.

Chapter 7

Unifying and Boosting Gradient-based Training-Free Neural Architecture Search

This chapter is based on the following work:
Shu, Y., Dai, Z. Wu, Z. & Low, B.K.H. (2022). Unifying and Boosting Gradient-based Training-Free Neural Architecture Search. Under review of *NeurIPS-22*.

7.1 Introduction

Recent years have witnessed a surging interest in applying *deep neural networks* (DNNs) in real-world applications, e.g., machine translation (Stahlberg, 2020), object detection (Zhao et al., 2019), among others. In order to achieve compelling performances in these applications, many domain-specific neural architectures have been handcrafted by human experts with considerable efforts. However, these efforts have gradually become unaffordable due to the growing demand for customizing

neural architectures for different tasks. To this end, *neural architecture search* (NAS) (Zoph and Le, 2017) has been proposed to design neural architectures automatically. While many *training-based* NAS algorithms (Pham et al., 2018; Liu et al., 2019) have achieved state-of-the-art (SOTA) performances in various tasks, their search costs usually are unaffordable in resource-constrained scenarios mainly due to their requirement for training DNNs during the search. As a result, a number of training-free metrics have been developed to realize *training-free NAS* (Mellor et al., 2020b; Chen et al., 2021). Surprisingly, these training-free NAS algorithms are able to achieve competitive empirical performances even compared with other training-based NAS algorithms while incurring significantly reduced search costs. Moreover, the architectures selected by these training-free NAS algorithms have been empirically found to transfer well to different tasks (Chen et al., 2021; Shu et al., 2021).

Despite the impressive empirical performances of the NAS algorithms using training-free metrics, *a unified theoretical analysis of these training-free metrics* is still lacking in the literature, leading to a few significant implications. Firstly, the theoretical relationships of these training-free metrics are unclear, making it challenging to explain *why they usually lead to comparable empirical results* (Abdelfattah et al., 2021). Secondly, there is no theoretical guarantee for the empirically observed compelling performances of the architectures selected by NAS algorithms using these training-free metrics. As a consequence, the reason *why NAS using these training-free metrics works well* is still not well understood, and hence there lacks theoretical assurances for NAS practitioners when deploying these algorithms. To the best of our knowledge, the theoretical aspect of NAS with training-free metrics has only been preliminarily studied by Shu et al. (2021). However, their analyses are only based on the training

rather than generalization performances of different architectures and are restricted to a *single* training-free metric. Thirdly, there may exist untapped potential in existing training-free NAS algorithms, which probably can be unveiled through a unified theoretical understanding of their training-free metrics.

To this end, we perform a unified theoretical analysis of *gradient-based* training-free NAS to resolve all the three problems discussed above in this work. Firstly, we theoretically prove the connections among different gradient-based training-free metrics in Sec. 7.3.1. Secondly, based on these provable connections, we derive a generalization bound for Deep Neural Networks (DNNs) with these training-free metrics and then use them to provide principled interpretations for the compelling empirical performances of existing training-free NAS algorithms (Secs. 7.3.2 and 7.3.3). Moreover, we demonstrate that our theoretical interpretation for training-free NAS algorithms, surprisingly, displays the same preference of architecture topology (i.e., wide or deep) as training-based NAS algorithms under certain conditions (Sec. 7.3.5), which helps to justify the practicality of our theoretical interpretations. Thirdly, by exploiting our unified theoretical analysis, we develop a novel NAS framework named *hybrid NAS* (HNAS) to consistently boost existing training-free NAS algorithms (Sec. 7.4) in a principled way. Remarkably, through a theory-inspired combination with *Bayesian optimization* (BO), our HNAS framework enjoys the advantages of both training-based (i.e., remarkable search effectiveness) and training-free (i.e., superior search efficiency) NAS simultaneously, making it more advanced than existing training-free and training-based NAS algorithms. Lastly, we use extensive experiments to verify the insights derived from our unified theoretical analysis, as well as the search effectiveness and efficiency of our non-trivial HNAS

framework (Sec. 7.5).

7.2 Notations and Backgrounds

7.2.1 Neural Tangent Kernel

To simplify the analysis in this work, we consider a L-layer DNN with identical widths $n_1 = \cdots = n_{L-1} = n$ and scalar output (i.e., $n_L = 1$) based on the formulation of DNNs in (Jacot et al., 2018). Let $f(\boldsymbol{x}, \boldsymbol{\theta})$ be the output of a DNN with input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ that are initialized using the standard normal distribution, the NTK matrix $\boldsymbol{\Theta} \in \mathbb{R}^{m \times m}$ over a dataset of size m is defined as

$$\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}', \boldsymbol{\theta}) \,. \tag{7.1}$$

Jacot et al. (2018) have shown that this NTK matrix $\boldsymbol{\Theta}$ will finally converge to a deterministic form $\boldsymbol{\Theta}_\infty$ in the infinitely wide DNN model. Meanwhile, Arora et al. (2019a); Allen-Zhu et al. (2019) have further proven that a similar result, i.e., $\boldsymbol{\Theta} \approx \boldsymbol{\Theta}_\infty$, can also be achieved in over-parameterized DNNs of finite width. Besides, Arora et al. (2019a); Lee et al. (2019a) have revealed that the training dynamics of DNNs can be well-characterized using this NTK matrix at initialization (i.e., $\boldsymbol{\Theta}_0$ based on the initialized model parameters $\boldsymbol{\theta}_0$) under certain conditions. More recently, Yang and Littwin (2021) have further demonstrated that these conclusions about NTK matrix shall also hold for DNNs with any reasonable architecture, even including recurrent neural networks (RNNs) and graph neural networks (GNNs). Therefore, the conclusions drawn based on the formulation above in this work shall also be applied to the NAS search spaces with complex architectures, which we validate empirically.

7.2.2 Gradient-based Training-free Metrics for NAS

We firstly introduce the training-free metrics that have been applied in training-free NAS. Particularly, in this work, we focus on the study of the training-free metrics that are based on the gradients of initialized model parameters, namely *gradient-based* training-free metrics as introduced below.

Gradient norm of initialized model parameters. While Abdelfattah et al. (2021) are the first to apply the gradient norm of initialized model parameters to estimate the generalization performance of candidate architectures in the search space, the same gradient norm has also been derived by Shu et al. (2021) for their approximation purpose. Following the notations in Sec. 7.2, let $\ell(\cdot,\cdot)$ be the loss function, we define this gradient norm over dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ as

$$\mathcal{M}_{\text{Grad}} \triangleq \left\| \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}_0} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right\|_2 . \tag{7.2}$$

SNIP and GraSP. While SNIP (Lee et al., 2019b) and GraSP (Wang et al., 2020a) are originally applied in training-free network pruning, Abdelfattah et al. (2021) borrow them into training-free NAS to estimate the performance of candidate architectures without the necessity of model training. Following the notations in Sec. 7.2, let $\mathbf{H}_i$ denote the hessian matrix induced by input $\boldsymbol{x}_i$, training-free metric SNIP and GraSP

on dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ can be defined as

$$\begin{aligned} \mathcal{M}_{\text{SNIP}} &\triangleq \left| \frac{1}{m} \sum_{i}^{m} \boldsymbol{\theta}_0^\top \nabla_{\boldsymbol{\theta}_0} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right| \\ \mathcal{M}_{\text{GraSP}} &\triangleq \left| \frac{1}{m} \sum_{i}^{m} \boldsymbol{\theta}_0^\top \left(\mathbf{H}_i \nabla_{\boldsymbol{\theta}_0} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right) \right| . \end{aligned} \tag{7.3}$$

Trace norm of NTK matrix at initialization. Recently, Shu et al. (2021) have reformulated the NAS problem into a constrained optimization using the trace norm of NTK matrix at initialization. Empirical results in (Shu et al., 2021) show that this NTK trace norm is highly correlated to the generalization performance of candidate architectures under their derived constraint. Let $\boldsymbol{\Theta}_0$ be the NTK matrix based on initialized model parameters $\boldsymbol{\theta}_0$, without considering the constraint in (Shu et al., 2021), we frame this training-free metric on dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ as blow using the notations in Sec. 7.2,

$$\mathcal{M}_{\text{Trace}} \triangleq \sqrt{\|\boldsymbol{\Theta}_0\|_{\text{tr}}/m} . \tag{7.4}$$

7.3 Theoretical Analyses for Training-free NAS

7.3.1 Connections among Training-free Metrics

Notably, though the gradient-based training-free metrics introduced in Sec. 7.2.2 seem to have distinct mathematical forms, most of them will actually achieve similar empirical performances in practice (Abdelfattah et al., 2021). More interestingly, these metrics in fact share the similarity of using the gradients of initialized model parameters in their calculations. Based on these facts, we propose the following hypothesis to explain the similar performances achieved by different training-free

metrics in Sec. 7.2.2: *The training-free metrics in Sec. 7.2.2 may be theoretically connected and hence could provide similar characterization for the generalization performances of architectures*. We validate this hypothesis affirmatively and use the following theorem to establish the theoretical connections among these metrics.

Theorem 7.1. *Let the loss function $\ell(\cdot,\cdot)$ in gradient-based training-free metrics be β-Lipschitz continuous and γ-Lipschitz smooth in the first argument. There exist the constants $C_1, C_2, C_3 > 0$, with a high probability,*

$$\mathcal{M}_{\text{Grad}} \leq C_1 \mathcal{M}_{\text{Trace}},\ \mathcal{M}_{\text{SNIP}} \leq C_2 \mathcal{M}_{\text{Trace}},\ \mathcal{M}_{\text{GraSP}} \leq C_3 \mathcal{M}_{\text{Trace}}\ .$$

The proof of Theorem 7.1 are given in Appendix D.1.1. Notably, our Theorem 7.1 implies that with a high probability, architectures of larger $\mathcal{M}_{\text{Grad}}$, $\mathcal{M}_{\text{SNIP}}$ or $\mathcal{M}_{\text{Grad}}$ will also achieve a larger $\mathcal{M}_{\text{Trace}}$ given the inequalities above. That is, the value of $\mathcal{M}_{\text{Grad}}$, $\mathcal{M}_{\text{SNIP}}$ and $\mathcal{M}_{\text{Grad}}$ for different architectures in the NAS search space should be highly correlated with the value of $\mathcal{M}_{\text{Trace}}$. As a consequence, these training-free metrics should be able to provide similar estimation of the generalization performances of architectures (validated in Sec. 7.5.2) and hence similar performances can be achieved when using these metrics (validated in Sec. 7.5.4). Overall, the training-free NAS metrics from Sec. 7.2.2 can all be theoretically connected with $\mathcal{M}_{\text{Trace}}$ despite their distinct mathematical forms. Note that though our Theorem 7.1 is only able to establish the theoretical connections between $\mathcal{M}_{\text{Trace}}$ and other training-free metrics, our empirical results in Sec. 7.6.1 further reveal that any two training-free metrics from Sec. 7.2.2 will also be highly correlated. Interestingly, these results serve as principled justifications for the similar performances achieved by these training-free metrics in (Abdelfattah et al., 2021).

7.3.2 A Generalization Bound Induced by Training-free Metrics

Let dataset $S=\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ be randomly sampled from a distribution $\mathcal{D}$, we denote $\mathcal{L}_S(\cdot)$ as the training error on S and $\mathcal{L}_{\mathcal{D}}(\cdot)$ as the corresponding *generalization error* on $\mathcal{D}$. Intuitively, a smaller generalization error indicates a better *generalization performance*. Thanks to the common theoretical underpinnings of gradient-based training-free metrics formalized by Theorem 7.1, we can perform a *unified generalization analysis* for DNNs in terms of these metrics by making use of the NTK theory (Jacot et al., 2018). Define $\ell(f, y) \triangleq (f-y)^2/2$ and $\eta_0 \triangleq \min\{2n^{-1}(\lambda_{\min}(\boldsymbol{\Theta}_\infty) + \lambda_{\max}(\boldsymbol{\Theta}_\infty))^{-1}, m\lambda_{\max}^{-1}(\boldsymbol{\Theta}_0)\}$ with $\lambda_{\min}(\cdot), \lambda_{\max}(\cdot)$ being the minimum and maximum eigenvalue of a matrix respectively, we derive the following theorem:

Theorem 7.2. *Assume* $\|\boldsymbol{x}_i\|_2 \leq 1$ *and* $f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), \lambda_{\min}(\boldsymbol{\Theta}_0), y_i \in [0,1]$ *for any* $(\boldsymbol{x}_i, y_i) \in S$. *There exists a constant* $N \in \mathbb{N}$ *such that for any* $n > N$, *when applying gradient descent with learning rate* $\eta < \eta_0$, *the generalization error of* f_t *at time* $t > 0$ *can be bounded as below with a high probability,*

$$\mathcal{L}_{\mathcal{D}}(f_t) \leq \mathcal{L}_S(f_t) + \mathcal{O}(\kappa/\mathcal{M}) \,.$$

Here, $\mathcal{M}$ *can be any metric in Sec. 7.2.2 and* $\kappa \triangleq \lambda_{\max}(\boldsymbol{\Theta}_0)/\lambda_{\min}(\boldsymbol{\Theta}_0)$ *is the condition number of* $\boldsymbol{\Theta}_0$.

Its proof is in Appendix D.1.2 and the second term $\mathcal{O}(\kappa/\mathcal{M})$ in Theorem 7.2 represents the *generalization gap* of DNN models. Notably, our Theorem 7.2 provides an explicit theoretical connection between the gradient-based training-free metrics from Sec. 7.2.2 and the generalization gap of DNNs, which later serves as the foundation to theoretically

interpret the compelling performances achieved by existing training-free NAS algorithms (Sec. 7.3.3). Compared to the traditional Rademacher complexity (Mohri et al., 2012), these training-free metrics provide alternative methods to measure the complexity of DNNs when estimating the generalization gap of DNNs.

7.3.3 Concrete Generalization Guarantees for Training-Free NAS

Since the first term $\mathcal{L}_S(\cdot)$ in our Theorem 7.2 may also depend on the training-free metric $\mathcal{M}$, it also needs to be taken into account when analyzing the generalization performance (or generalization error $\mathcal{L}_{\mathcal{D}}(\cdot)$) for training-free NAS methods. To this end, in this section, we derive concrete generalization guarantees for NAS methods using training-free metrics by considering two different scenarios (i.e., the *realizable* and *non-realizable* scenarios) for the training error term $\mathcal{L}_S(\cdot)$ in Theorem 7.2, which finally give rise to principled interpretations for different training-free NAS methods (Abdelfattah et al., 2021; Shu et al., 2021; Chen et al., 2021).

The realizable scenario. Similar to (Mohri et al., 2012), we assume that a zero training error (i.e., $\mathcal{L}_S(f_t) \to 0$ when t is sufficiently large) can be achieved in the realizable scenario. By further assuming that the condition number κ in Theorem 7.2 is bounded by κ_0 for all candidate architectures in the search space, we can then derive the following generalization guarantee (Corollary 7.1) for the realizable scenario.

Corollary 7.1. *Under the conditions in Theorem 7.2, for f_t at convergence (i.e., $t \to \infty$) in the realizable scenario and for any training-free metric $\mathcal{M}$*

from Sec. 7.2.2, the following holds with a high probability,

$$\mathcal{L}_{\mathcal{D}}(f_t) \leq \mathcal{O}(1/\mathcal{M}) \,.$$

Corollary 7.1 is obtained by introducing $\mathcal{L}_S(f_t) = 0$ and $\kappa \leq \kappa_0$ into Theorem 7.2. Importantly, Corollary 7.1 suggests that in the realizable scenario, the generalization error of DNNs is negatively correlated with the metrics from Sec. 7.2.2. That is, an architecture with a larger value of training-free metric $\mathcal{M}$ generally achieves a better generalization performance. This implies that in order to select well-performing architectures, we can simply maximize $\mathcal{M}$ to find $\mathcal{A}^* = \arg\max_{\mathcal{A}} \mathcal{M}(\mathcal{A})$ where $\mathcal{A}$ denotes any architecture in the search space. Interestingly, this formulation aligns with the training-free NAS method from (Abdelfattah et al., 2021), which has made use of the metrics $\mathcal{M}_{\text{Grad}}, \mathcal{M}_{\text{SNIP}}$ and $\mathcal{M}_{\text{GraSP}}$ to achieve good empirical performances. Therefore, our Corollary 7.1 provides a valid generalization guarantee and also a principled justification for the method from (Abdelfattah et al., 2021).

The non-realizable scenario. In practice, different candidate architectures in a NAS search space typically have diverse non-zero training errors (Shu et al., 2021) and κ (Chen et al., 2021). Therefore, the assumptions of the zero training error and the bounded κ in the realizable scenario above may be impractical. In light of this, we drop these two assumptions and derive the following generalization guarantee (Corollary 7.2) for the non-realizable scenario, which, interestingly, facilitates theoretically grounded interpretations for the training-free NAS methods from (Shu et al., 2021; Chen et al., 2021).

Corollary 7.2. *Under the conditions in Theorem 7.2, for any f_t at time $t > 0$*

and any training-free metric $\mathcal{M}$ from Sec. 7.2.2 in the non-realizable scenario, there exists a constant $c > 0$ such that with a high probability,

$$\mathcal{L}_{\mathcal{D}}(f_t) \leq \frac{1}{2} m \left(1 - \eta \mathcal{M}^2/(mc)\right)^{2t} + \mathcal{O}(\kappa/\mathcal{M}) \,.$$

Its proof is given in Appendix D.1.3. Notably, our Corollary 7.2 suggests that when $\mathcal{M} \in [0, \sqrt{mc/\eta}]$, an architecture with a larger value of the metric $\mathcal{M}$ will lead to a better generalization performance because such a model has both a faster convergence (i.e., the first term decreases faster w.r.t time t) and a smaller generalization gap (i.e., the second term is smaller). Interestingly, Shu et al. (2021) have leveraged this insight to introduce the training-free metric of $\mathcal{M}_{\text{Trace}}$ with a constraint, which has achieved a higher correlation with the generalization performance of architectures than the metrics from (Abdelfattah et al., 2021). This therefore implies that our Corollary 7.2 followed by (Shu et al., 2021) provides a better characterization of the generalization performance of architectures than Corollary 7.1 followed by (Abdelfattah et al., 2021) since the non-realizable scenario we have considered will be more realistic than the realizable scenario as explained above. Meanwhile, Corollary 7.2 also suggests that there exists a trade-off in terms of $\mathcal{M}$ between the model convergence (i.e., the first term) and the generalization gap (i.e., the second term) when $\mathcal{M} > \sqrt{mc/\eta}$, which surprisingly is similar to the empirically motivated trainability and expressivity trade-off in (Chen et al., 2021). In addition, Corollary 7.2 also indicates that for architectures achieving similar values of $\mathcal{M}$, the ones with smaller condition numbers κ generally achieve better generalization performance. Interestingly, such a result also aligns with the conclusion from (Chen et al., 2021). Therefore, our Corollary 7.2 also provides a principled justification for

the training-free NAS method in (Chen et al., 2021).

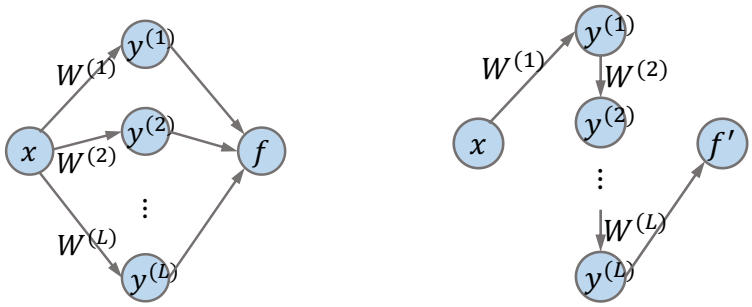
7.3.4 Guaranteed Transferability for Training-Free NAS

In practice, the transferability of the architectures selected by both training-based and training-free NAS algorithms has been widely verified empirically (Liu et al., 2019; Chen et al., 2021; Shu et al., 2021). This naturally begs the question: *Can the transferability of the architectures selected by training-free NAS algorithms also be theoretically guaranteed?* To answer this question, we perform a unified analysis of the transferability of the architectures selected by NAS algorithms using the training-free metrics from Sec. 7.2.2. Let $\mathcal{M}$ and $\mathcal{M}'$ be any the same training-free metric from Sec. 7.2.2 that are evaluated on dataset S (sampled from an underlying distribution $\mathcal{D}$) and S' (sampled from a different underlying $\mathcal{D}'$) with empirical distribution $P(\boldsymbol{x})$ and $P'(\boldsymbol{x})$, respectively. Define $\mathcal{M}_{\min} \triangleq \min(\mathcal{M}_{\text{Trace}}, \mathcal{M}'_{\text{Trace}})$ and $Z \triangleq \int \|P(\boldsymbol{x}) - P'(\boldsymbol{x})\| \mathrm{d}\boldsymbol{x}$, we then derive the following theorem:

Theorem 7.3. *Assume that the 1-Lipschitz continuous activation function σ in the DNN model f satisfies $|\sigma(x)| \leq |x|$, under the conditions in Theorem 7.2, there exists a constant $\alpha > 0$ such that as $n \to \infty$, for any f_t obtained at time $t > 0$, the following holds with a high probability when $\mathcal{M}' > \alpha(2n_0\mathcal{M}_{\min})^{-1}Z)$,*

$$\mathcal{L}_{\mathcal{D}}(f_t) \leq \mathcal{L}_S(f_t) + \mathcal{O}(\kappa/(\mathcal{M}' - \alpha(2n_0\mathcal{M}_{\min})^{-1}Z)) \,.$$

Theorem 7.3 reveals that the generalization error w.r.t. an underlying distribution $\mathcal{D}$ can be upper-bounded by the combination of the training error on its empirical dataset S (i.e., $\mathcal{L}_S(\cdot)$) and *a generalization gap which depends on the metric $\mathcal{M}'$ evaluated using a different dataset S'*. This suggests that an architecture selected using the training-free metric on a dataset



(a) Wide architecture (b) Deep architecture

Figure 7.1: Two different architecture topologies for our analysis.

S' is also likely to produce compelling performances on another dataset S which may follow a different underlying distribution $\mathcal{D}$. Therefore, Theorem 7.3 provides a theoretical foundation for the transferability of the architectures selected by NAS methods using the training-free metrics from Sec. 7.2.2. Similar to the conclusions from (Shu et al., 2021), Theorem 7.3 demonstrates that a smaller $(2n_0)^{-1}Z$ leads to a better transferability for the architectures selected by training-free NAS because it reduces the generalization gap in Theorem 7.3. This transferability is improved when either (a) the input dimension (i.e., n_0) is large which is satisfied by many real-world datasets (e.g., ImageNet (Deng et al., 2009)) and (b) the divergence between the datasets (i.e., Z) is small.

7.3.5 Connection to Architecture Topology

Interestingly, we can even prove that the condition number κ in our Corollary 7.2 is theoretically related to the architecture topology, i.e., whether the architecture is wide (and shallow) or deep (and narrow). Inspired by the analysis from (Shu et al., 2020), we firstly analyze the eigenvalues of the NTK matrices of two different architecture topologies (i.e., wide vs. deep architectures), which gives us an insight into the difference between their corresponding κ. We mainly consider the following wide (i.e., f)

and deep (i.e., f') architecture illustrated in Figure 7.1, respectively:

$$f(\boldsymbol{x}) = \mathbf{1}^\top \textstyle\sum_{i=1}^{L} \mathbf{W}^{(i)} \boldsymbol{x} \ , \ f'(\boldsymbol{x}) = \mathbf{1}^\top (\textstyle\prod_{i=1}^{L} \mathbf{W}^{(i)}) \boldsymbol{x} \tag{7.5}$$

where $\mathbf{W}^{(i)} \in \mathbb{R}^{n \times n}$ for any $i \in \{1, \cdots, L\}$ and every element of $\mathbf{W}^{(i)}$ is independently initialized using the standard normal distribution. Here, $\mathbf{1}$ denotes an n-dimensional vector with every element being one. Let $\boldsymbol{\Theta}_0$ and $\boldsymbol{\Theta}_0'$ be the NTK matrices of f and f' that are evaluated on the finite dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, respectively, we derive the following theorem:

Theorem 7.4. *Let dataset S be normalized using its statistic mean and covariance such that $\mathbb{E}[\boldsymbol{x}] = 0$ and $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$ given $\mathbf{X} \triangleq [\boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_m]$, we have*

$$\boldsymbol{\Theta}_0 = Ln \cdot \mathbf{I} \ , \ \mathbb{E}\left[\boldsymbol{\Theta}_0'\right] = Ln^L \cdot \mathbf{I} \ .$$

Its proof is in Appendix D.1.5. Notably, Theorem 7.4 shows that the NTK matrix of the wide architecture in (7.5) is guaranteed to be a scaled identity matrix, whereas the NTK matrix of the deep architecture in (7.5) is a scaled identity matrix *only in expectation* over random initialization. Consequently, we always have $\kappa = 1$ for the initialized wide architecture, while $\kappa > 1$ with high probability for the initialized deep architecture. Also note that as we have discussed in Sec. 7.3.3, our Corollary 7.2 shows that (given similar values of $\mathcal{M}$) an architecture with a smaller κ is likely to generalize better. Therefore, this implies that wide architectures generally achieve better generalization performance than deep architectures (given similar values of $\mathcal{M}$). This, surprisingly, aligns with the findings from (Shu et al., 2020) which shows that wide architectures are preferred in *training-based NAS* due to their competitive performances in practice. More interestingly, based on the definition of $\mathcal{M}_{\text{Trace}}$ (7.4), Theorem 7.4 also indicates that deep architectures are expected to have larger values

of $\mathcal{M}_{\text{Trace}}$ (due to the larger scale of $\mathbb{E}\left[\boldsymbol{\Theta}_0'\right]$ for deep architectures) and hence achieve larger model complexities than wide architectures.

7.4 Hybrid Neural Architecture Search

7.4.1 A Unified Objective for Training-Free NAS

Our theoretical understanding of training-free NAS in Sec. 7.3 finally allows us to address the following question in a principled way: *How can we consistently boost existing training-free NAS algorithms?* Specifically, to realize this target, we propose to select well-performing architectures by minimizing the upper bound on the generalization error in Corollary 7.2 given any training-free metric from Sec. 7.2.2. We expect this choice to lead to improved performances over the method from (Abdelfattah et al., 2021) because Corollary 7.2 provides a more practical generalization guarantee for training-free NAS than Corollary 7.1 followed by (Abdelfattah et al., 2021) (Sec. 7.3.3). Formally, let $\mathcal{A}$ be any architecture in the search space and let $\mathcal{M}$ be any training-free metric from Sec. 7.2.2, then the NAS problem can be formulated below in a unified manner:

$$\min_{\mathcal{A}} \frac{1}{2} m \left(1 - \frac{\eta}{mc} \mathcal{M}^2(\mathcal{A})\right)^{2t} + \mathcal{O}\left(\frac{\kappa(\mathcal{A})}{\mathcal{M}(\mathcal{A})}\right). \tag{7.6}$$

We further reformulate (7.6) into the following form:

$$\min_{\mathcal{A}} \kappa(\mathcal{A})/\mathcal{M}(\mathcal{A}) + \mu F(\mathcal{M}^2(\mathcal{A}) - \nu) \tag{7.7}$$

where $F(x) \triangleq x^{2t}$, and μ and ν are hyperparameters we introduced to absorb the impact of all other parameters in (7.6). Compared with the diverse form of NAS objectives in (Abdelfattah et al., 2021; Chen et al.,

Algorithm 7.1 Hybrid Neural Architecture Search (HNAS)

1: **Input:** Training and validation dataset, metric $\mathcal{M}$ evaluated on architecture pool $\mathcal{P}$, and $F(\cdot)$
2: **for** step $k = 1, \ldots, K$ **do**
3: Choose μ_k, ν_k using BO algorithm based on its exploration and exploitation trade-off
4: Obtain the optimal candidate $\mathcal{A}_k^*$ in $\mathcal{P}$ by solving (7.7)
5: Evaluate the validation performance of $\mathcal{A}_k^*$
6: Update the GP surrogate applied in the BO algorithm
7: Select the final architecture $\mathcal{A}^*$ achieving the best validation performance among $\{\mathcal{A}_k^*\}_{k=1}^K$.

2021; Shu et al., 2021), our (7.7) proposes a unified form of NAS objectives for all the training-free metrics in Sec. 7.2.2, making it easier for practitioners to deploy NAS with different training-free metrics. Note that our NAS objective in (7.7) is a natural consequence of our generalization guarantee in Corollary 7.2 and therefore will be more theoretically grounded, in contrast to the empirically motivated objective in (Chen et al., 2021). Moreover, our (7.7) advances the training-free NAS method based on $\mathcal{M}_{\text{Trace}}$ from (Shu et al., 2021), because our (7.7) *(a)* is derived from the generalization error instead of the training error (that is followed by (Shu et al., 2021)) of DNNs, which therefore will be more sound and practical, *(b)* have unified all the training-free metrics from Sec. 7.2.2, and *(c)* have considered the impact of condition number κ which is shown to be critical in practice (see our Sec. 7.6.2). Above all, our unified NAS objective in (7.7) is expected to be able to lead to improved performances over other existing training-free NAS methods.

7.4.2 Optimization and Search Algorithm

Our theoretically motivated NAS objective in (7.7) has unified all training-free metrics from Sec. 7.2.2 and improved over existing training-free NAS methods. However, its practical deployment requires the de-

termination of the hyperparameters μ and ν,[1] which can be non-trivial in practice. To this end, we further introduce *Bayesian optimization* (BO) (Snoek et al., 2012) to optimize the hyperparameters μ and ν in order *to maximize the true validation performance* of the architectures selected by different μ and ν. Specifically, BO uses a Gaussian process (GP) as a surrogate to model the objective function (i.e., the validation performance) in order to sequentially choose the queried inputs (i.e., the values of μ and ν). This finally completes our theoretically grounded NAS framework called *hybrid NAS* (HNAS), which novelly unifies all training-free metrics from Sec. 7.2.2 (Algorithm 7.1). In every iteration k of HNAS, we firstly select the optimal candidate $\mathcal{A}_k^*$ by maximizing our training-free NAS objective in (7.7) using the values of μ and ν queried by BO in the current iteration. Next, we evaluate the validation performance of $\mathcal{A}_k^*$ and then use it to update the GP surrogate, which will be used to choose the values of μ and ν in the next iteration. After HNAS completes, the final selected architecture is chosen as the one achieving the best validation performance among all the optimal candidates. Thanks to the use of validation performance as the objective for BO, our HNAS is expected to be able to enjoy the advantages of both training-free (i.e., superior search efficiency) and training-based NAS (i.e., remarkable search effectiveness) as supported by our extensive empirical results in Sec. 7.5.4. More interestingly, by novelly introducing BO to optimize the low-dimensional continuous hyperparameters μ and ν rather than the high-dimensional discrete architectural hyperparameters in the NAS search space, our HNAS is able to avoid the high-dimensional discrete optimization issues that standard BO algorithms usually attain when directly optimizing in the NAS search space, allowing our HNAS to be more efficient and effective in practice as

[1] We usually set $t = 1$ in practice.

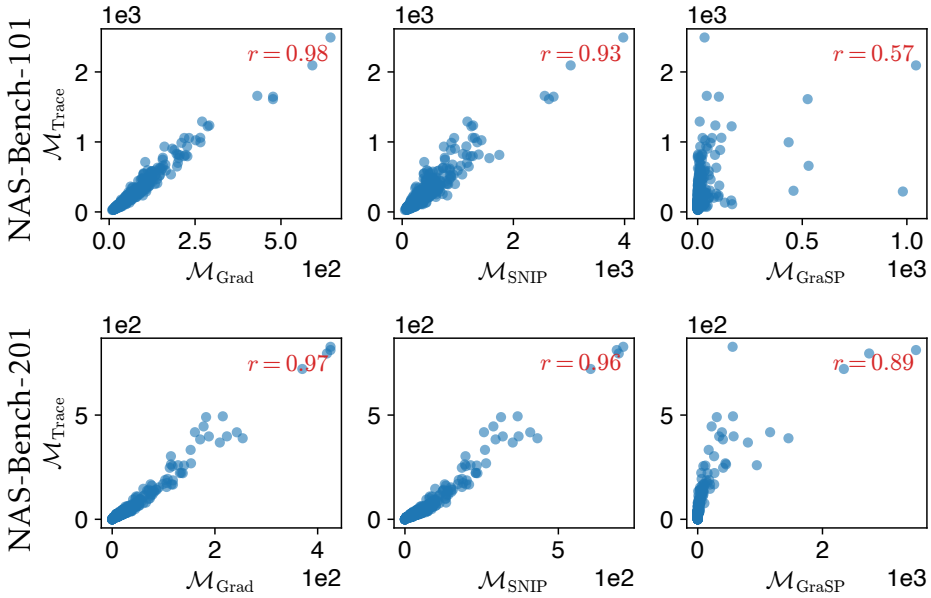


Figure 7.2: Correlation between $\mathcal{M}_{\text{Trace}}$ and other training-free metrics from Sec. 7.2.2, which are evaluated in NAS-Bench-101/201. The correlation coefficient r is given in the corner of each plot.

supported in our Sec. 7.5.4.

7.5 Experiments

7.5.1 Connections among Training-free Metrics

We firstly verify the theoretical connections between $\mathcal{M}_{\text{Trace}}$ and other training-free metrics from Sec. 7.2.2 by examining their Spearman correlations for all architectures in NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-201 (Dong and Yang, 2020) using CIFAR-10 (Krizhevsky et al., 2009). Figure 7.2 illustrates the result where all these training-free metric are evaluated using a batch of randomly sampled data following that of (Abdelfattah et al., 2021). [2] Notably, Figure 7.2 demonstrates that $\mathcal{M}_{\text{Trace}}$ and other training-free metrics from Sec. 7.2.2 are indeed highly correlated since they consistently achieve high positive corre-

[2] We follow the same approach to evaluate these training-free metrics in the following sections.

Table 7.1: Connection between the generalization guarantees in our Sec.7.3.3 and the generalization performance (test error on CIFAR-10) of architectures in NAS-Bench-101 and NAS-Bench-201.

Metric	NAS-Bench-101		NAS-Bench-201	
	Spearman	Kendall's Tau	Spearman	Kendall's Tau
Realizable scenario				
$\mathcal{M}_{\text{Grad}}$	−0.25	−0.17	0.64	0.47
$\mathcal{M}_{\text{SNIP}}$	−0.21	−0.15	0.64	0.47
$\mathcal{M}_{\text{GraSP}}$	−0.45	−0.31	0.57	0.40
$\mathcal{M}_{\text{Trace}}$	−0.30	−0.21	0.54	0.39
Non-realizable scenario				
$\mathcal{M}_{\text{Grad}}$	0.35	0.23	0.75	0.56
$\mathcal{M}_{\text{SNIP}}$	0.37	0.25	0.75	0.56
$\mathcal{M}_{\text{GraSP}}$	0.46	0.32	0.69	0.50
$\mathcal{M}_{\text{Trace}}$	0.33	0.23	0.70	0.51

lations. These results thus strongly support our theoretical results in Theorem 7.1. The correlation between any two training-free metrics from Sec. 7.2.2 is in Sec. 7.6.1, which shows that all training-free metrics from Sec. 7.2.2 are also highly correlated.

7.5.2 Generalization Guarantees for Training-Free NAS

We then verify the validity of our generalization guarantees for training-free NAS (Sec. 7.3.3) by examining the correlation between the generalization bound in the realizable (Corollary 7.1) or non-realizable (Corollary 7.2) scenario and the test errors of architectures in NAS-Bench-101/201. Similar to HNAS (Algorithm 7.1), we use BO with hundreds of iterations to decide the non-trivial parameters in Corollary 7.2. Table 7.1 summarizes the results on CIFAR-10 where a higher positive correlation implies a better agreement between our generalization guarantee and the generalization performance of architectures. As shown, the generalization bound in the realizable scenario performs a compelling characterization

Table 7.2: Comparison of topology, κ, and $\mathcal{M}_{\text{Trace}}$ of different architectures. The topology width/depth of an architecture is followed by the maximum value in the search space (separated by a slash).

Architecture	**Topology**		$\mathcal{M}_{\text{Trace}}$	**Condition Number**
	Width	Depth		κ
NASNet	5.0/5.0	2/6	31±2	118±41
AmoebaNet	4.0/5.0	4/6	36±2	110±39
ENAS	5.0/5.0	2/6	36±2	98±33
DARTS	3.5/4.0	3/5	33±2	122±58
SNAS	4.0/4.0	2/5	31±2	126±47
WIDE	4.0/4.0	2/5	27±1	141±36
DEEP	**1.5**/4.0	**5**/5	**131±16**	**209±107**

of the test errors in NAS-Bench-201 by achieving large positive correlations, whereas it fails to provide a precise characterization in a larger search space, i.e., NAS-Bench-101. Fortunately, our generalization bound in the non-realizable scenario performs consistent improvement over it by obtaining higher positive correlations. These results thus imply that Corollary 7.1 may only provide a good characterization for training-free NAS in small-scale search spaces, whereas our Corollary 7.2 can be more valid and robust in practice. As a result, our (7.6) following Corollary 7.2 should indeed be able to improve over the NAS objective in (Abdelfattah et al., 2021) following Corollary 7.1 as justified in our Sec. 7.4 above. Meanwhile, the comparable results achieved by all training-free metrics from Sec. 7.2.2 further validate the connections among these metrics (Theorem 7.1). Moreover, the results in Sec. 7.6.2 further confirm the validity and practicality of our generalization guarantees for training-free NAS.

7.5.3 Connection to Architecture Topology

To validate the theoretical connections between architecture topology (wide vs. deep) and the value of training-free metric $\mathcal{M}_{\text{Trace}}$ as well as the condition number κ shown in Sec. 7.3.5, we compare the topology width/depth, $\mathcal{M}_{\text{Trace}}$ and κ of the architectures selected by different state-of-the-art training-based NAS algorithms in the DARTS search space, including NASNet (Zoph et al., 2018), AmoebaNet (Real et al., 2019b), ENAS (Pham et al., 2018), DARTS (Liu et al., 2019), and SNAS (Xie et al., 2019b). Table 7.2 summarizes the results where we apply the same definition of topology width/depth in (Shu et al., 2020) (more details in (Shu et al., 2020)). We also include the widest (called WIDE) and the deepest (called DEEP) architecture in the DARTS search space into this comparison. As shown, wide architectures (i.e., all architectures except DEEP in Table 7.2) consistently achieve lower condition number κ and smaller values of $\mathcal{M}_{\text{Trace}}$ than deep architecture (i.e., DEEP), which aligns with our theoretical insights in Sec. 7.3.5.

7.5.4 Effectiveness and Efficiency of HNAS

To justify that our theoretically motivated HNAS is able to enjoy the advantages of both training-free (i,e., the superior search efficiency) and training-based (i.e., the remarkable search effectiveness) NAS, we compare it with other baselines in NAS-Bench-201 (Table 7.3). HNAS, surprisingly, advances both training-based and training-free baselines by consistently selecting architectures achieving the best performances, leading to smaller gaps toward the optimal test errors in the search space. Meanwhile, HNAS requires at most 13× lower search costs than training-based NAS algorithms, which is even smaller than the training-free baseline

Table 7.3: Comparison among NAS algorithms in NAS-Bench-201. The result of HNAS is reported with the mean and standard deviation of 5 independent searches and its search costs are evaluated on a Nvidia 1080Ti. C & D in the last column denote continuous and discrete search space, respectively.

Architecture	Test Accuracy (%)			Cost (GPU Sec.)	Method	Applicable Space
	C10	C100	IN-16			
ResNet (He et al., 2016)	93.97	70.86	43.63	-	manual	-
REA†	93.92±0.30	71.84±0.99	45.15±0.89	12000	evolution	C & D
RS (w/o sharing)†	93.70±0.36	71.04±1.07	44.57±1.25	12000	random	C & D
REINFORCE†	93.85±0.37	71.71±1.09	45.24±1.18	12000	RL	C & D
BOHB†	93.61±0.52	70.85±1.28	44.42±1.49	12000	BO+bandit	C & D
ENAS‡ (Pham et al., 2018)	93.76±0.00	71.11±0.00	41.44±0.00	15120	RL	C
DARTS (1st)‡ (Liu et al., 2019)	54.30±0.00	15.61±0.00	16.32±0.00	16281	gradient	C
DARTS (2nd)‡ (Liu et al., 2019)	54.30±0.00	15.61±0.00	16.32±0.00	43277	gradient	C
GDAS‡ (Dong and Yang, 2019b)	93.44±0.06	70.61±0.21	42.23±0.25	8640	gradient	C
NASWOT (Mellor et al., 2020b)	92.96±0.81	69.98±1.22	44.44±2.10	306	training-free	C & D
TE-NAS (Chen et al., 2021)	93.90±0.47	71.24±0.56	42.38±0.46	1558	training-free	C
KNAS (Xu et al., 2021)	93.05	68.91	34.11	4200	training-free	C & D
NASI (Shu et al., 2021)	93.55±0.10	71.20±0.14	44.84±1.41	120	training-free	C
HNAS ($\mathcal{M}_{\text{Grad}}$)	**94.04**±0.21	71.75±1.04	**45.91**±0.88	3010	hybrid	C & D
HNAS ($\mathcal{M}_{\text{SNIP}}$)	**93.94**±0.02	71.49±0.11	**46.07**±0.14	2976	hybrid	C & D
HNAS ($\mathcal{M}_{\text{GraSP}}$)	**94.13**±0.13	**72.59**±0.82	**46.24**±0.38	3148	hybrid	C & D
HNAS ($\mathcal{M}_{\text{Trace}}$)	**94.07**±0.10	**72.30**±0.70	**45.93**±0.37	3006	hybrid	C & D
Optimal	94.37	73.51	47.31	-	-	-

† Reported by Dong and Yang (2020).
‡ Re-evaluated for 5 times using the codes provided by Dong and Yang (2020).

KNAS. Moreover, thanks to the superior evaluation efficiency of training-free metrics, HNAS can be deployed efficiently in not only continuous (where search space is represented as a supernet) but also discrete search space. As for NAS under limited search budgets (Figure 7.3), HNAS also advances all other baselines by achieving improved search efficiency and effectiveness. Sec. 7.6.4 further includes the impressive search results achieved by HNAS on CIFAR-10/100 and ImageNet in the DARTS search space. Overall, HNAS is indeed able to enjoy the advantages of both training-free (i.e., the superior search efficiency) and training-based NAS (i.e., the remarkable search effectiveness), which consistently boosts existing training-free NAS methods.

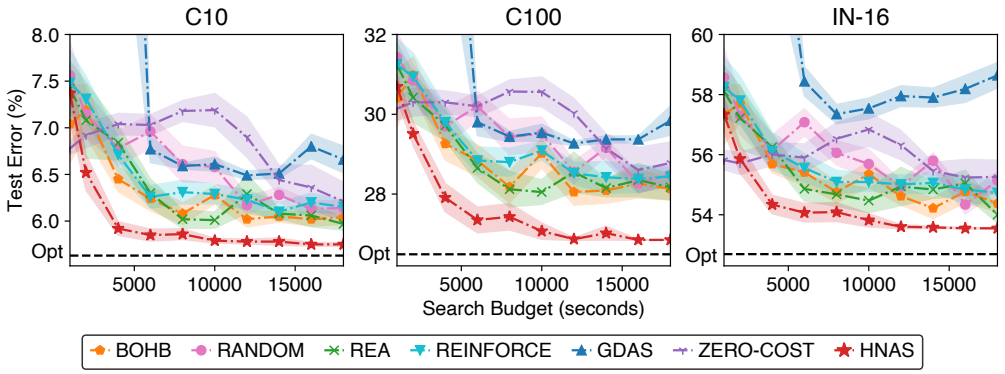


Figure 7.3: Comparison between HNAS ($\mathcal{M}_{\text{Trace}}$) and other NAS baselines in NAS-Bench-201 under varying search budgets. Here, the ZERO-COST method is borrowed from (Abdelfattah et al., 2021) by using $\mathcal{M}_{\text{Trace}}$ and each algorithm is reported with the mean and standard error of ten independent searches.

Table 7.4: Connection between any two training-free metrics (i.e., $\mathcal{M}_1$ and $\mathcal{M}_2$ in the table) from Sec. 7.2.2 in NAS-Bench-101/201. Note that each training-free metric is evaluated using a batch of randomly sampled data from CIFAR-10 following that of (Abdelfattah et al., 2021).

$\mathcal{M}_1$	$\mathcal{M}_2$	NAS-Bench-101			NAS-Bench-201		
		Pearson	Spearman	Kendall's Tau	Pearson	Spearman	Kendall's Tau
Gradient-based training-free metrics							
$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	0.98	0.98	0.87	1.00	1.00	0.97
$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{GraSP}}$	0.35	0.61	0.43	0.60	0.92	0.77
$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{Trace}}$	0.98	0.98	0.87	0.98	0.97	0.85
$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	0.34	0.59	0.42	0.55	0.92	0.77
$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{Trace}}$	0.94	0.93	0.77	0.97	0.96	0.83
$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$	0.37	0.57	0.40	0.69	0.89	0.73
$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{Grad}}$	0.95	0.96	0.83	0.88	0.94	0.80
$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{SNIP}}$	0.91	0.92	0.75	0.87	0.94	0.78
$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{GraSP}}$	0.37	0.65	0.46	0.45	0.87	0.69
$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{Trace}}$	0.96	0.96	0.84	0.89	0.97	0.86
Non-gradient-based training-free metrics							
$\mathcal{M}_{\text{Fisher}}$	$\mathcal{M}_{\text{Trace}}$	0.69	0.97	0.85	0.30	0.78	0.69
$\mathcal{M}_{\text{SynFlow}}$	$\mathcal{M}_{\text{Trace}}$	0.02	0.50	0.34	0.07	0.49	0.35
$\mathcal{M}_{\text{NASWOT}}$	$\mathcal{M}_{\text{Trace}}$	0.08	0.11	0.08	0.10	0.32	0.22

7.6 More Results

7.6.1 Connections among Training-Free Metrics

Besides the theoretical (Theorem 7.1) and empirical (Sec. 7.3.1) connections between $\mathcal{M}_{\text{Trace}}$ and other gradient-based training-free metrics from Sec. 7.2.2, we further show in Table 7.4 that any two metrics from Sec. 7.2.2 are highly correlated, i.e., they consistently achieve large posi-

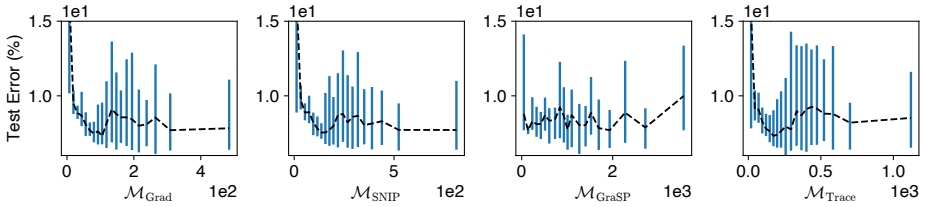
tive correlations in both NAS-Bench-101 and NAS-Bench-201. Similar to the results in our Sec. 7.3.1, the correlation between $\mathcal{M}_{\text{GraSP}}$ and any other training-free metric is generally lower than other pairs, which may result from the hessian matrix that has only been applied in $\mathcal{M}_{\text{GraSP}}$. Furthermore, to figure out *whether our Theorem 7.1 is also applicable to non-gradient-based training-free metrics*, we then provide the correlation between $\mathcal{M}_{\text{Fisher}}$ (Turner et al., 2020), $\mathcal{M}_{\text{SynFlow}}$ (Tanaka et al., 2020), $\mathcal{M}_{\text{NASWOT}}$ (Mellor et al., 2020b) and $\mathcal{M}_{\text{Trace}}$ for a comparison. [3] Interestingly, both $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ achieve higher positive correlations with $\mathcal{M}_{\text{Trace}}$ than $\mathcal{M}_{\text{NASWOT}}$ in general. According to their mathematical forms in the corresponding papers, such a phenomenon may result from the fact that $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ have contained certain gradient information while $\mathcal{M}_{\text{NASWOT}}$ only relies on the outputs of each layer in an initialized architecture. [4] These results therefore imply that our Theorem 7.1 may also provide valid theoretical connections for the training-free metrics that are not gradient-based but still contain certain gradient information.

7.6.2 Valid Generalization Guarantees for Training-Free NAS

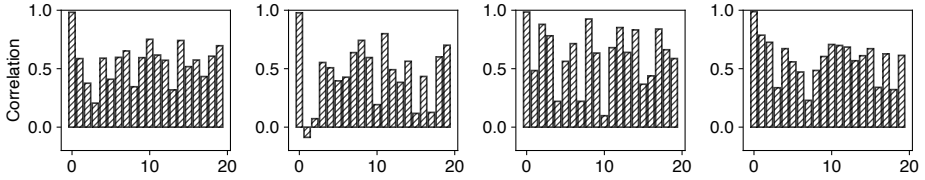
To further support that our Corollary 7.2 presents a more practical and valid generalization guarantee for training-free NAS in practice, we examine the true generalization performances of all candidate architectures under their different value of training-free metrics in Figure 7.4 (a) and exhibit the correlation between the condition number and the true generalization performances of all candidate architectures in Figure 7.4 (b).

[3] Note that both $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ are also adopted by Abdelfattah et al. (2021) for training-free NAS.

[4] Since the gradient information contained in $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ is different from the gradient of initialized model parameters from loss function or the output of a DNN, they are not taken as the gradient-based training-free metrics in this work.



(a) Varying architecture performances in the search space



(b) Correlation between condition numbers and architecture performances

Figure 7.4: (a) Varying architecture performances under different value of training-free metrics in NAS-Bench-201. Note that the x-axis denotes the averaged value of training-free metrics over the architectures grouped in the same bin and y-axis denoted the test error evaluated on CIFAR-10. (b) Correlation between the condition numbers and the true generalization performances of the architectures within the same bin (i.e., the y-axis). Note that the x-axis denotes the corresponding 20 bins in Figure 7.4 (a).

Specifically, we group the value of training-free metrics in NAS-Bench-201 into 20 bins and then plot the test errors on CIFAR-10 of all candidate architectures within the same bin into the blue vertical lines in Figure 7.4 (a). Besides, we plot the averaged test errors over the architectures within the same bin into the black dash lines in Figure 7.4 (a). Besides, each correlation between condition number and test error in Figure 7.4 (b) is computed using the candidate architectures within the same bin.

Notably, as illustrated by the black dash lines in Figure 7.4 (a), there consistently exists a trade-off for all the training-free metrics in Sec. 7.2.2. Specifically, there exists an optimal value $\mathcal{M}_{\text{opt}}$ for each training-free metric $\mathcal{M}$ that is capable of achieving the best generalization performance in the search space. When $\mathcal{M} < \mathcal{M}_{\text{opt}}$, architecture with a larger value of $\mathcal{M}$ typically enjoys a better generalization performance. On the contrary, when $\mathcal{M} > \mathcal{M}_{\text{opt}}$, architecture with a smaller value of $\mathcal{M}$ generally

achieves a better generalization performance. Interestingly, these results perfectly align with our Corollary 7.2. Furthermore, Figure 7.4 (b) shows that the condition number is indeed highly correlated to the generalization performance of candidate architectures and a smaller condition number is generally preferred in order to select well-performing architectures in training-free NAS. More interestingly, similar phenomenons can also be found in (Shu et al., 2021) and (Chen et al., 2021). Remarkably, our Corollary 7.2 can provide theoretically grounded interpretations for these results, whereas Corollary 7.1 fails to characterize these phenomenons. Consequently, our Corollary 7.2 is shown to be more practical and valid in practice.

Based on the conclusions above, we then compare the impacts of the trade-off and condition number κ mentioned above by examining the correlation between the true generalization performances of candidate architectures and their training-free metrics applied in different scenarios. Here, we use the same parameters applied in Sec. 7.5.2 for Corollary 7.2. Table 7.5 summarizes the comparison. Note that the non-realizable scenario is equivalent to the realizable scenario + trade-off + κ as suggested by our Corollary 7.2. As revealed in Table 7.5, both trade-off and condition number κ are necessary to achieve an improved characterization of architecture performances over the one in the realizable scenario followed by (Abdelfattah et al., 2021), which again verifies the practicality and validity of our Corollary 7.2. More interestingly, condition number κ is shown to be more essential than the trade-off for training-free NAS in order to improve the correlations in the realizable scenario. By integrating both trade-off and condition number κ into the realizable scenario, the non-realizable scenario consistently enjoys the highest correlations on different datasets, which also further verifies the improvement of our

Table 7.5: Correlation between the test errors of candidate architectures in NAS-Bench-201 and their training-free metrics applied in several different scenarios. We refer to Sec. 7.3.3 for more details about the trade-off and condition number κ applied in the following scenarios.

Dataset	Scenario	Spearman				Kendall's Tau			
		$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$
C10	Realizable	0.637	0.639	0.566	0.538	0.469	0.472	0.400	0.387
	Realizable + Trade-off	0.642	0.641	0.570	0.549	0.475	0.474	0.403	0.397
	Realizable + κ	0.724	0.728	0.658	0.657	0.530	0.533	0.474	0.474
	Non-realizable	**0.750**	**0.748**	**0.686**	**0.697**	**0.559**	**0.556**	**0.501**	**0.512**
C100	Realizable	0.638	0.638	0.571	0.535	0.473	0.475	0.409	0.385
	Realizable + Trade-off	0.642	0.645	0.578	0.546	0.476	0.481	0.414	0.394
	Realizable + κ	0.716	0.719	0.649	0.651	0.527	0.529	0.469	0.470
	Non-realizable	**0.740**	**0.746**	**0.680**	**0.686**	**0.552**	**0.557**	**0.498**	**0.504**
IN-16	Realizable	0.578	0.578	0.550	0.486	0.430	0.433	0.397	0.354
	Realizable + Trade-off	0.588	0.589	0.566	0.526	0.438	0.441	0.408	0.382
	Realizable + κ	0.646	0.649	0.612	0.587	0.472	0.474	0.443	0.423
	Non-realizable	**0.682**	**0.685**	**0.655**	**0.660**	**0.505**	**0.506**	**0.480**	**0.482**

training-free NAS objective (7.7) over the one used in (Abdelfattah et al., 2021).

7.6.3 Guaranteed Transferability for Training-Free NAS

To verify our transferability guarantee for training-free NAS (Sec. 7.3.4), we examine the deviation of the correlation between the architecture performance and the generalization bounds in Sec. 7.3.3 using training-free metrics evaluated on different datasets. That is, training-free metrics and architecture performance can be evaluated on different datasets. Table 7.6 summarizes the results using CIFAR-10/100 (C10/100) and ImageNet-16-120 (IN-16) (Chrabaszcz et al., 2017) in NAS-Bench-201 where we use the same parameters as Sec. 7.5.2 for Corollary 7.2. Notably, nearly the same correlations (i.e., with extremely small deviations) are achieved for training-free metrics evaluated on different datasets. This implies that the training-free metrics computed on a dataset S can also provide a good characterization of the architecture performance evaluated on another dataset S'. Therefore, the architectures selected by training-free NAS algorithms on S are also likely to produce a compelling performance on S'.

Table 7.6: Deviation of the correlation between the test errors in NAS-Bench-201 and the generalization bounds in Sec. 7.3.3 using training-free metrics evaluated on various datasets. Each correlation is reported with the mean and standard deviation using the metrics evaluated on CIFAR-10/100 and ImageNet-16-120.

Dataset	**Training-free Metrics**			
	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$
Realizable scenario				
C10	0.64±0.01	0.64±0.01	0.58±0.02	0.55±0.01
C100	0.64±0.01	0.64±0.01	0.58±0.03	0.54±0.02
IN-16	0.57±0.01	0.57±0.01	0.52±0.03	0.47±0.02
Non-realizable scenario				
C10	0.75±0.00	0.75±0.00	0.69±0.01	0.69±0.00
C100	0.74±0.00	0.74±0.00	0.69±0.01	0.69±0.01
IN-16	0.69±0.00	0.69±0.00	0.63±0.01	0.65±0.00

That is, the transferability of the architectures selected by training-free NAS is guaranteed.

7.6.4 HNAS in the DARTS Search Space

To support the effectiveness and efficiency of our HNAS, we also apply HNAS in the DARTS (Liu et al., 2019) search space to find well-performing architectures on CIFAR-10/100 and ImageNet (Deng et al., 2009). Specifically, we sample a pool of 60000 architecture to evaluate their training-free metrics on CIFAR-10 in order to maintain high computational efficiency for these training-free metrics. For the results on CIFAR-10/100, we then apply the BO algorithm for 25 steps with a 10-epoch model training for the selected architectures in our HNAS (Algorithm 7.1). As for the results on ImageNet, we apply the BO algorithm for 10 steps with a 3-epoch model training for the selected architectures in our HNAS. We follow (Liu et al., 2019) to construct 20-layer final selected architectures with an auxiliary tower of weight 0.4 for CIFAR-10

(0.6 for CIFAR-100) located at 13-th layer and 36 initial channels. We evaluate these architectures on CIFAR-10/100 using stochastic gradient descent (SGD) of 600 epochs with a learning rate cosine scheduled from 0.025 to 0 for CIFAR-10 (from 0.035 to 0.001 for CIFAR-100), momentum 0.9, weight decay $3\times10^{-4}$and batch size 96. Both Cutout (Devries and Taylor, 2017), and ScheduledDropPath linearly increased from 0 to 0.2 for CIFAR-10 (from 0 to 0.3 for CIFAR-100) are employed for regularization purposes on CIFAR-10/100. As for the evaluation on ImageNet, we train the 14-layer architecture from scratch for 250 epochs with a batch size of 1024. The learning rate is warmed up to 0.7 for the first 5 epochs and then decreased to zero with a cosine schedule. We adopt the SGD optimizer with 0.9 momentum and a weight decay of 3×10^{-5}.

The results on CIFAR-10/100 and ImageNet are summarized in Table 7.7 and Table 7.8, respectively. As shown in Table 7.7, both our HNAS (C10) and HNAS (C100) are capable of achieving state-of-the-art performance on CIFAR-10 and CIFAR-100, correspondingly, while incurring lower search costs than other training-based NAS algorithms. Even compared with other training-free NAS baselines, e.g., TE-NAS, our HNAS can still enjoy a compelling search cost. Overall, these results further validate that our HNAS is indeed able to enjoy the superior search efficiency of training-free NAS and also the remarkable search effectiveness of training-based NAS. More interestingly, our HNAS (C10) can achieve a lower test error on CIFAR-10 but a higher test error on CIFAR-100 when compared with HNAS (C100). This result indicates that similar to training-based NAS algorithms, directly searching on the target dataset is also able to improve the final performance in HNAS. By exploiting this advantage over other training-free NAS baselines, our HNAS thus is capable of selecting architectures achieving higher performances, as shown

Table 7.7: Performance comparison among state-of-the-art (SOTA) neural architectures on CIFAR-10/100. The performance of the final architectures selected by HNAS is reported with the mean and standard deviation of five independent evaluations. The search costs are evaluated on a single Nvidia 1080Ti. Note that HNAS (C10 or C100) denoted the architecture selected by our HNAS using the dataset CIFAR-10 or CIFAR-100, respectively.

Architecture	Test Error (%)		Params (M)		Search Cost (GPU Hours)	Search Method
	C10	C100	C10	C100		
DenseNet-BC (Huang et al., 2017b)	3.46*	17.18*	25.6	25.6	-	manual
NASNet-A (Zoph et al., 2018)	2.65	-	3.3	-	48000	RL
AmoebaNet-A (Real et al., 2019b)	3.34±0.06	18.93†	3.2	3.1	75600	evolution
PNAS (Liu et al., 2018)	3.41±0.09	19.53*	3.2	3.2	5400	SMBO
ENAS (Pham et al., 2018)	2.89	19.43*	4.6	4.6	12	RL
NAONet (Luo et al., 2018a)	3.53	-	3.1	-	9.6	NAO
DARTS (2nd) (Liu et al., 2019)	2.76±0.09	17.54†	3.3	3.4	24	gradient
GDAS (Dong and Yang, 2019b)	2.93	18.38	3.4	3.4	7.2	gradient
NASP (Yao et al., 2020)	2.83±0.09	-	3.3	-	2.4	gradient
P-DARTS (Chen et al., 2019)	2.50	-	3.4	-	7.2	gradient
DARTS- (avg) (Chu et al., 2020)	2.59±0.08	17.51±0.25	3.5	3.3	9.6	gradient
SDARTS-ADV (Chen and Hsieh, 2020)	2.61±0.02	-	3.3	-	31.2	gradient
R-DARTS (L2) (Zela et al., 2020a)	2.95±0.21	18.01±0.26	-	-	38.4	gradient
TE-NAS♯ (Chen et al., 2021)	2.83±0.06	17.42±0.56	3.8	3.9	1.2	training-free
NASI-ADA Shu et al. (2021)	2.90±0.13	16.84±0.40	3.7	3.8	0.24	training-free
HNAS (C10)	2.62±0.04	17.10±0.18	3.4	3.5	2.4	hybrid
HNAS (C100)	2.78±0.05	**16.29**±0.14	3.7	3.8	2.7	hybrid

† Reported by Dong and Yang (2019b) with their experimental settings.
* Obtained by training corresponding architectures without cutout (Devries and Taylor, 2017) augmentation.
♯ Reported by Shu et al. (2021) with their experimental settings.

in Table 7.7. Similar results are also achieved on ImageNet as shown in Table 7.8. Overall, these results have further supported the superior search efficiency and remarkable search effectiveness of our HNAS that we have verified in Sec. 7.5.4.

7.6.5 Ablation Studies

Ablation study on initialization method. While our theoretical analyses throughout this work are based on the initialization using the standard normal distribution (Sec. 7.2), [5] we wonder *whether our theoretical results are also applicable to DNNs using different initialization methods*, e.g., Xavier (Glorot and Bengio, 2010) and Kaiming (He et al., 2015a) initialization. Specifically, we compare the correlation between the true generalization

[5] Note that this initialization is equivalent to the LeCun initialization (LeCun et al., 2012) according to (Jacot et al., 2018).

Table 7.8: Performance comparison among SOTA image classifiers on ImageNet.

Architecture	Test Error (%)		Params (M)	+× (M)	Search Cost (GPU Days)
	Top-1	Top-5			
Inception-v1 (Szegedy et al., 2015a)	30.1	10.1	6.6	1448	-
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	-
ShuffleNet 2×(v2) (Ma et al., 2018)	25.1	7.6	7.4	591	-
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000
AmoebaNet-A (Real et al., 2019b)	25.5	8.0	5.1	555	3150
PNAS (Liu et al., 2018)	25.8	8.1	5.1	588	225
MnasNet-92 (Tan et al., 2019b)	25.2	8.0	4.4	388	-
DARTS (Liu et al., 2019)	26.7	8.7	4.7	574	4.0
SNAS (mild) (Xie et al., 2019b)	27.3	9.2	4.3	522	1.5
GDAS (Dong and Yang, 2019b)	26.0	8.5	5.3	581	0.21
ProxylessNAS (Cai et al., 2019b)	24.9	7.5	7.1	465	8.3
DARTS- (Chu et al., 2020)	23.8	7.0	4.5	467	4.5
SDARTS-ADV (Chen and Hsieh, 2020)	25.2	7.8	5.4	594	1.3
TE-NAS (C10) (Chen et al., 2021)	26.2	8.3	5.0	-	0.05
TE-NAS (ImageNet) (Chen et al., 2021)	24.5	7.5	5.4	-	0.17
NASI-ADA (Shu et al., 2021)	25.0	7.8	4.9	559	0.01
HNAS (C100)	24.8	7.8	5.2	601	0.1
HNAS (ImageNet)	24.3	7.4	5.1	575	0.5

performances of all candidate architectures in NAS-Bench-201 and the generalization guarantees in Sec. 7.3.3 that are evaluated using different initialization methods. Table 7.9 summarizes the comparison. Here, we use the same parameters applied in Sec. 7.5.2 for Corollary 7.2. Notably, Table 7.9 shows that our generalization guarantees for training-free NAS, i.e., Corollary 7.1, 7.2, can also perform well for training-free NAS using DNNs initialized with different methods, indicating a wider application of our generalization guarantees in Sec. 7.3.3. Of note, LeCun initialization can achieve the best results among the three initialization methods in Table 7.9 since it satisfies our assumption about the initialization of DNNs. As an implication, LeCun initialization is more preferred when using the training-free metrics from Sec. 7.2.2 to characterize the architecture performances in training-free NAS.

Table 7.9: Correlation between the test errors (on CIFAR-10) of all architectures in NAS-Bench-201 and our generalization guarantees in Sec. 7.3.3 that are evaluated on DNNs using different initialization methods.

Initialization	Spearman				Kendall's Tau			
	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$
Realizable scenario								
LeCun (LeCun et al., 2012)	0.637	0.639	0.566	0.538	0.469	0.472	0.400	0.387
Xavier (Glorot and Bengio, 2010)	0.608	0.627	0.449	0.465	0.445	0.463	0.316	0.334
He (He et al., 2015a)	0.609	0.615	0.340	0.460	0.446	0.454	0.242	0.334
Non-realizable scenario								
LeCun (LeCun et al., 2012)	0.750	0.748	0.686	0.697	0.559	0.556	0.501	0.512
Xavier (Glorot and Bengio, 2010)	0.676	0.685	0.615	0.635	0.493	0.501	0.442	0.460
He (He et al., 2015a)	0.607	0.611	0.505	0.569	0.436	0.439	0.358	0.407

Ablation study on batch size. Theoretically, the training-free metrics from Sec. 7.2.2 are defined over the whole training dataset. In practice, we usually only apply a batch of randomly sampled data points to evaluate these training-free metrics in order to achieve a desirable computational efficiency, which follows (Abdelfattah et al., 2021). To investigate the impact of batch size on these metrics, we examine the correlation between the true generalization performances of all candidate architectures in NAS-Bench-201 and their generalization guarantees in the non-realizable scenario under varying batch sizes. Table 7.10 summarizes the results. Here, we use the same parameters applied in Sec. 7.5.2 for Corollary 7.2. Besides the impact of batch size on training-free metrics, we also include the impact of batch size on condition number κ in this table. Specifically, in the upper part of Table 7.10, the correlations are evaluated using a batch size of 64 for κ and varying batch sizes for any training-free metric $\mathcal{M}$ from Sec. 7.2.2. Meanwhile, in the lower part of Table 7.10, the correlations are evaluated using varying batch sizes for κ and a batch size of 64 for any training-free metric $\mathcal{M}$. Notably, Table 7.10 shows that similar results will be achieved even when training-free metrics are evaluated under varying batch sizes, whereas κ evaluated under varying batch sizes will lead to distinguished results, indicating that κ is more

Table 7.10: Correlation between the test errors (on CIFAR-10) of all architectures in NAS-Bench-201 and their generalization guarantees in the non-realizable scenario under varying batch size.

Batch Size	Spearman				Kendall's Tau			
	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$
Batch size 64 for κ and varying batch sizes for any $\mathcal{M}$								
4	0.737	0.741	0.671	0.684	0.547	0.550	0.487	0.501
8	0.739	0.743	0.676	0.689	0.549	0.552	0.492	0.506
16	0.747	0.748	0.685	0.690	0.556	0.556	0.499	0.507
32	0.750	0.748	0.687	0.690	0.558	0.556	0.502	0.506
64	0.750	0.748	0.686	0.697	0.559	0.556	0.501	0.512
Varying batch sizes for κ and batch size 64 for any $\mathcal{M}$								
4	0.578	0.585	0.569	0.509	0.416	0.421	0.402	0.362
8	0.597	0.603	0.591	0.542	0.429	0.433	0.419	0.386
16	0.628	0.633	0.620	0.582	0.462	0.455	0.442	0.414
32	0.663	0.666	0.645	0.621	0.479	0.481	0.462	0.445
64	0.750	0.748	0.686	0.697	0.559	0.556	0.501	0.512

sensitive to batch size than training-free metrics. As an implication, while a small batch size is also able to perform well in practice, a large batch size is more preferred when using our generalization guarantees for training-free NAS.

Ablation study on layer width. While our theoretical analyses are based on over-parameterized DNNs, i.e., $n > N$ in our Theorem 7.2, we are also curious about *how the layer width will influence our empirical results*. In particular, we examine the correlation between the true generalization performances of all candidate architectures in NAS-Bench-201 and their generalization guarantee in the non-realizable scenario under varying layer width. Similar to the ablation study on batch size, we investigate the impacts of layer width on the training-free metrics from Sec. 7.2.2 and the condition number κ separately. Table 7.11 summarizes the results. Here, we use the same parameters applied in Sec. 7.5.2 for Corollary 7.2. As shown in Table 7.11, our generalization guarantee in the non-realizable scenario also performs well when layer width becomes

Table 7.11: Correlation between the test errors (on CIFAR-10) of all architectures in NAS-Bench-201 and their generalization guarantees in the non-realizable scenario under varying layer widths, which are measured by the number of initial channels in our experiments. Larger initial channels indicates a large layer width.

Init Channels	Spearman				Kendall's Tau			
	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$	$\mathcal{M}_{\text{Grad}}$	$\mathcal{M}_{\text{SNIP}}$	$\mathcal{M}_{\text{GraSP}}$	$\mathcal{M}_{\text{Trace}}$
4 channels for κ and varying channels for any $\mathcal{M}$								
4	0.744	0.746	0.688	0.732	0.550	0.552	0.499	0.539
8	0.750	0.753	0.707	0.744	0.556	0.559	0.515	0.550
16	0.753	0.753	0.728	0.750	0.558	0.559	0.535	0.556
32	0.755	0.756	0.736	0.752	0.560	0.562	0.543	0.558
Varying channels for κ and 32 channels for any $\mathcal{M}$								
4	0.755	0.756	0.736	0.752	0.560	0.562	0.543	0.558
8	0.720	0.722	0.700	0.709	0.529	0.531	0.512	0.522
16	0.698	0.700	0.677	0.681	0.511	0.514	0.492	0.498
32	0.686	0.688	0.664	0.664	0.501	0.503	0.481	0.484

smaller. Surprisingly, similar results can be achieved for training-free metrics evaluated under varying layer widths, whereas a larger layer width for training-free metrics typically leads to marginally higher correlations in Table 7.11. On the contrary, a larger layer width for κ leads to lower correlations in Table 7.11. This may result from the similar behavior that can be achieved by layer width and topology width since both layer width and topology width are used to measure the width of DNN but in totally different perspectives. Therefore, increasing layer width will make deep architectures (in terms of topology) more indistinguishable from wide architectures (in terms of topology) and hence make it harder to apply our generalization guarantee in Corollary 7.2 to characterize the architecture performances in a search space. As an implication, a large layer width for training-free metrics and a smaller layer width for condition number κ are more preferred when applying our generalization guarantees for training-free NAS in practice.

Ablation study on generalization guarantees, transferability guarantee and HNAS algorithm using non-gradient-based training-free metrics. Since Sec. 7.6.1 has validated that our Theorem 7.1 may also provide valid theoretical connections for certain non-gradient-based training-free metrics, we wonder *whether our theoretical generalization guarantees, transferability guarantee and HNAS based on Theorem 7.1 are also applicable to these non-gradient-based training-free metrics*. In particular, we firstly examine the correlation between the true generalization performances of all candidate architectures in NAS-Bench-201 and their generalization (Sec. 7.3.3) and transferability guarantees (Sec. 7.3.4) using training-free metrics $\mathcal{M}_{\text{Fisher}}$, $\mathcal{M}_{\text{SynFlow}}$ and $\mathcal{M}_{\text{NASWOT}}$. Table 7.12 summarizes the results. Here, we use the same parameters applied in Sec. 7.5.2 for Corollary 7.2. While $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ enjoy higher correlation to $\mathcal{M}_{\text{Trace}}$ than $\mathcal{M}_{\text{NASWOT}}$ in Sec. 7.6.1, our generalization and transferability guarantees also performs better when using $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$. We then apply our HNAS based on these training-free metrics in NAS-Bench-201 and the Table 7.13 summarizes the search results. Similarly, our HNAS based on $\mathcal{M}_{\text{Fisher}}$ and $\mathcal{M}_{\text{SynFlow}}$ can also find better-performing architectures than HNAS ($\mathcal{M}_{\text{NASWOT}}$). Surprisingly, HNAS ($\mathcal{M}_{\text{SynFlow}}$) can even achieve competitive results when compared with HNAS using gradient-based training-free metrics. These results therefore indicate that our HNAS sometimes may also be able to improve over training-free NAS using non-gradient-based training-free metrics especially when these non-gradient-based training-free metrics contain certain gradient information.

Ablation study on the optimization process of HNAS. In this section, we examine the evolution of the correlation between the test errors of

Table 7.12: Correlation between the test errors of all architectures in NAS-Bench-201 and our generalization guarantees in Sec. 7.3.3 using training-free metrics $\mathcal{M}_{\text{KNAS}}$, $\mathcal{M}_{\text{Fisher}}$, $\mathcal{M}_{\text{SynFlow}}$ and $\mathcal{M}_{\text{NASWOT}}$ that are evaluated on various datasets. Each correlation is reported with the mean and standard deviation using the metrics evaluated on CIFAR-10/100 and ImageNet-16-120.

Dataset	Spearman				Kendall's Tau			
	$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{Fisher}}$	$\mathcal{M}_{\text{SynFlow}}$	$\mathcal{M}_{\text{NASWOT}}$	$\mathcal{M}_{\text{KNAS}}$	$\mathcal{M}_{\text{Fisher}}$	$\mathcal{M}_{\text{SynFlow}}$	$\mathcal{M}_{\text{NASWOT}}$
Realizable scenario								
C10	0.53±0.02	0.39±0.01	0.78±0.00	0.09±0.02	0.39±0.02	0.29±0.01	0.58±0.00	0.10±0.00
C100	0.53±0.03	0.39±0.01	0.76±0.00	0.09±0.02	0.38±0.02	0.29±0.01	0.57±0.00	0.11±0.01
IN-16	0.46±0.02	0.32±0.01	0.75±0.00	0.16±0.02	0.33±0.02	0.24±0.01	0.56±0.00	0.15±0.02
Non-realizable scenario								
C10	0.66±0.02	0.51±0.00	0.81±0.00	0.05±0.00	0.49±0.02	0.37±0.00	0.61±0.00	0.03±0.00
C100	0.67±0.03	0.51±0.01	0.80±0.02	0.05±0.01	0.49±0.02	0.37±0.00	0.60±0.00	0.03±0.00
IN-16	0.62±0.04	0.44±0.00	0.78±0.00	0.05±0.01	0.45±0.03	0.32±0.00	0.59±0.00	0.03±0.00

candidate architectures in the NAS search space and their generalization guarantees in the non-realizable scenario with the BO steps in our HNAS framework. Figure 7.5 illustrates the results in NAS-Bench-201, which are based on CIFAR-10 dataset and training-free metric $\mathcal{M}_{\text{Trace}}$. Note that in every BO step of Figure 7.5, the Spearman correlation we reported corresponds to the pair of hyperparameters μ and ν that achieves the best validation performance in the query history. As shown in Figure 7.5, our HNAS framework, interestingly, is indeed selecting better-performing architectures by selecting hyperparameters μ and ν that can achieve higher Spearman correlation in the search space. These results therefore further justify the advantages of introducing BO algorithms into training-free NAS.

7.7 Conclusion and Discussion

This work performs a unified theoretical analysis of NAS algorithms using gradient-based training-free metrics, which allows us to *(a)* theoretically unveil the connections among these training-free metrics, *(b)* provide theoretical guarantees for the empirically observed compelling

Table 7.13: Comparison among HNAS using different training-free metrics in NAS-Bench-201. The performance of each HNAS variant is reported with the mean and standard deviation of five independent searches and the search costs are evaluated on a single Nvidia 1080Ti.

Architecture	**Test Accuracy (%)**			**Search Cost** (GPU Sec.)
	C10	C100	IN-16	
HNAS ($\mathcal{M}_{\text{Grad}}$)	94.04±0.21	71.75±1.04	45.91±0.88	3010
HNAS ($\mathcal{M}_{\text{SNIP}}$)	93.94±0.02	71.49±0.11	46.07±0.14	2976
HNAS ($\mathcal{M}_{\text{GraSP}}$)	94.13±0.13	72.59±0.82	46.24±0.38	3148
HNAS ($\mathcal{M}_{\text{Trace}}$)	94.07±0.10	72.30±0.70	45.93±0.37	3006
HNAS ($\mathcal{M}_{\text{KNAS}}$)	94.19±0.06	72.94±0.52	46.31±0.38	3081
HNAS ($\mathcal{M}_{\text{Fisher}}$)	93.28±0.73	69.42±1.36	42.85±2.09	3309
HNAS ($\mathcal{M}_{\text{SynFlow}}$)	94.13±0.00	72.50±0.00	45.47±0.00	3615
HNAS ($\mathcal{M}_{\text{NASWOT}}$)	92.10±0.62	66.81±0.32	39.26±0.72	2832
Optimal	94.37	73.51	47.31	-

performance of these training-free NAS algorithms, and *(c)* exploit these theoretical understandings to develop a novel framework called HNAS that can consistently boost existing training-free NAS. We expect that our theoretical understanding could provide valuable prior knowledge for the design of training-free metrics and NAS search space in the future. Moreover, considering the close relationship between network pruning and NAS, we expect our unified theoretical analyses to be capable of inspiring more theoretical understanding and improvement over existing training-free network pruning algorithms in the literature. More importantly, the impressive performance achieved by our HNAS framework is expected to be able to encourage more attention to the integration of training-free and training-based approaches in other machine learning fields in order to enjoy the advantages of these two types of methods simultaneously.

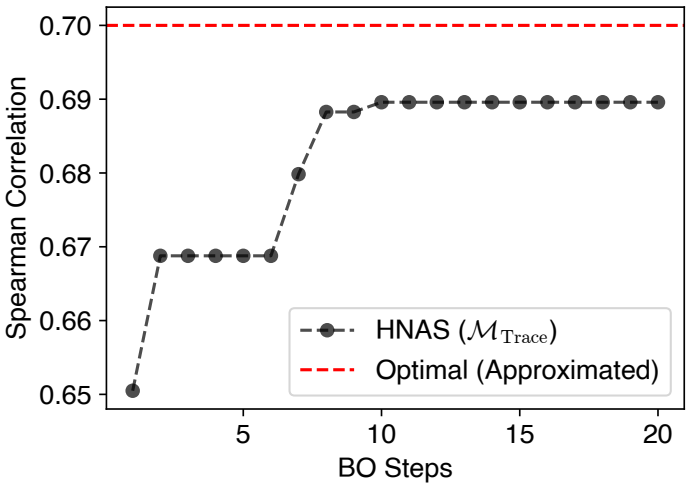


Figure 7.5: Evolution of the correlation between the test errors (on CIFAR-10) of all architectures in NAS-Bench-201 and their generalization guarantees (using $\mathcal{M}_{\text{Trace}}$) in the non-realizable scenario with the BO steps in our HNAS framework.

Chapter 8

Conclusion and Future Work

8.1 Summary

This thesis has presented our four works in NAS that help to understand and further improve the search effectiveness as well as the search efficiency of recent NAS algorithms, supported by large-scale experiments. To summarize this thesis, we give a brief summary of our four works in this thesis respectively as below.

Chapter 4. In the literature, most of the efforts have been dedicated to developing efficient and effective NAS algorithms (Akimoto et al., 2019; Liu et al., 2019; Luo et al., 2018b; Nayman et al., 2019; Xie et al., 2019b) in the NAS area. Nonetheless, to our best knowledge, less or even no attention has been devoted to those architectures selected by popular NAS algorithms to understand *what types of architectures are selected by these NAS algorithms and why they are selected*. These questions are however fundamental to understanding and improving existing NAS algorithms. In Chapter 4, we reveal that those architectures selected by popular NAS algorithms tend to favor wide and shallow cells by examining the connection topologies of these architectures, which may result from their

fast and stable convergence in the model training. We further empirically and theoretically show that those architectures with wider and shallower cells consistently enjoy a smoother loss landscape and smaller gradient variance than their random variants, which contribute to their better convergence behavior and consequently the selection of these architectures given a limited search budget. However, architectures with wide and shallow cells may not generalize better than other candidate architectures despite their well-behaved convergence, leading to further research on how to accurately estimate the final generalization performance of candidate architectures in the search space. Besides, the inclusion of our aforementioned understanding into the design of NAS search space may help to further improve the search efficiency and effectiveness of popular NAS algorithms.

Chapter 5. Conventional NAS algorithms aim to select only *one single architecture* from their search spaces and hence have overlooked the capability of other candidate architectures in helping improve the performance of their final selected architecture. Our work in Chapter 5 therefore presents two novel ensemble search algorithms, i.e., NESBS (MC sampling) and NESBS (SVGD-RD), that can effectively and efficiently select well-performing ensembles of architectures from the NAS search space, which can further improve the performance achieved by conventional NAS algorithms. Empirical results show that in both classification and adversarial defense tasks on various benchmark datasets, our NESBS algorithms are able to achieve consistently improved performances compared with conventional NAS algorithms while incurring comparable search costs. Furthermore, even compared with the ensemble search baselines (e.g., Deep Ensemble and ensemble using Monte Carlo Dropout),

our algorithms can also enjoy improved search effectiveness and efficiency. As our algorithms rely on the effectiveness of our ensemble performance estimation (Sec. 5.2.1), an interesting future direction is to study how to further improve this performance estimation in order to achieve a higher ensemble performance.

Chapter 6. Recent training-based NAS algorithms typically require the model training of the supernet or the candidate architectures in the search space, which is however computationally expensive in practice. This naturally begs the question *whether NAS can be conducted without any model training during the search process.* To this end, in Chapter 6, we propose a novel NAS algorithm called *NAS at Initialization* (NASI) by using the theory of *Neural Tangent Kernel* (NTK) (Jacot et al., 2018; Lee et al., 2019a) to formally estimate the converged performance of infinite-wide DNNs at initialization, hence allowing NAS to be conducted without any model training. More importantly, the architectures selected by NASI are proven to be transferable with its *label-* and *data-agnostic* search. Empirical results on various benchmarks have shown that our NASI algorithm significantly advances other popular NAS algorithms in search efficiency while maintaining a comparable generalization performance for those selected architectures. We think that our theoretical justification for the transferability of our NASI can be generalized to other popular NAS algorithms, which thus provides further theoretical understanding of NAS.

Chapter 7. Though recent training-free NAS algorithms using various zero-cost metrics are already shown to be able to achieve competitive performance even compared with those training-based counterparts, the reason *why NAS using these training-free metrics performs well* in

practice remains a mystery in the literature. To this end, our work in Chapter 7 presents a unified theoretical analysis based on the theory of NTK to derive the theoretical connection, the generalization guarantees and the transferability guarantees for gradient-based training-free NAS, which helps to understand the practical performance achieved by various training-free NAS algorithms. Remarkably, these theoretical analyses indeed align with the empirical results, which therefore implies the validity of our theoretical analyses. Moreover, by exploiting these theoretical understandings, a novel NAS framework called HNAS is proposed to further improve the search effectiveness of training-free NAS algorithms. Finally, we believe our principled analyses can inspire more theoretical studies in the field of NAS or even AutoML.

8.2 Future Outlook

8.2.1 NAS in Different Scenarios

While many well-performing NAS algorithms have been proposed in a standard setting i.e., NAS based on a one-shot supervised task in a single agent, NAS algorithms in other different scenarios are still underdeveloped. For example, while standard NAS algorithms usually require dataset labels to select best-performing architecture, these labels are expensive to get in real-world applications. In this scenario, *how can we select well-performing architectures without labels?* Besides, the real-world datasets are typically collected sequentially. This naturally leads to the question *how to select well-performing architectures in an online manner.* There has also been an increasing interest in introducing collaborative mechanism into machine learning algorithms for a higher effectiveness or efficiency in recent years (Sim et al., 2021). By borrowing this idea

into NAS, *how can we select well-performing architectures a collaborative NAS setting where several agents need to find their best-performing architectures on self-interested datasets collaboratively?* Above all, developing NAS algorithms in different Scenarios will be an interesting and promising direction to explore.

8.2.2 Beyond NAS in AutoML

As a sub-field of AutoML, NAS shares many similarities to other sub-fields, such as automatic data augmentation and hyper-parameter optimization. Inspired by the impressive results achieved by our works in this thesis, *how to extend our ideas in this thesis into automatic data augmentation and hyper-parameter optimization* will be an interesting topic to investigate on. Besides, algorithms are usually designed designed to find optimal strategies for automatic data augmentation, neural architecture search or hyper-parameter optimization separately. However, in practice, the final performance of real-world tasks are decided by these three aspects jointly. These separately designed algorithms thus may only be able to achieve a sub-optimal solution for the target task. Meanwhile, algorithms designed separately on automatic data augmentation, neural architecture search or hyper-parameter optimization are hard to be integrated together in practice especially when they are customized over specific tasks. For example, the same data augmentation and hyper-parameters are required in DARTS algorithm (Liu et al., 2019) in order to compare the performance of candidate architectures. Consequently, *how to develop an algorithm to select optimal strategies for automatic data augmentation, neural architecture search or hyper-parameter optimization jointly* will be important to make AutoML more applicable and also be able to achieve better performance in practice.

Bibliography

Abdelfattah, M. S., Mehrotra, A., Dudziak, L., and Lane, N. D. (2021). Zero-cost proxies for lightweight NAS. In *Proc. ICLR*.

Akimoto, Y., Shirakawa, S., Yoshinari, N., Uchida, K., Saito, S., and Nishida, K. (2019). Adaptive stochastic natural gradient method for one-shot neural architecture search. In *Proc. ICML*, volume 97, pages 171–180.

Allen-Zhu, Z., Li, Y., and Song, Z. (2019). A convergence theory for deep learning via over-parameterization. In *Proc. ICML*, pages 242–252.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., and Wang, R. (2019a). On exact computation with an infinitely wide neural net. In *Proc. NeurIPS*, pages 8139–8148.

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. (2019b). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *Proc. ICML*, pages 322–332.

Awasthi, P., Frank, N., and Mohri, M. (2020). On the rademacher complexity of linear hypothesis sets. arXiv:2007.11045.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18:153:1–153:43.

Beck, A. and Tetruashvili, L. (2013). On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23:2037–2060.

Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. V. (2018). Understanding and simplifying one-shot architecture search. In *Proc. ICML*, pages 549–558.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2018). SMASH: One-shot model architecture search through hypernetworks. In *Proc. ICLR*.

Cai, H., Zhu, L., and Han, S. (2019a). Proxylessnas: Direct neural architecture search on target task and hardware. In *Proc. ICLR*.

Cai, H., Zhu, L., and Han, S. (2019b). ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proc. ICLR*.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *Proc. IEEE S&P*, pages 39–57.

Chen, W., Gong, X., and Wang, Z. (2021). Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *Proc. ICLR*.

Chen, X. and Hsieh, C. (2020). Stabilizing differentiable architecture search via perturbation-based regularization. In *Proc. ICML*, pages 1554–1565.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proc. ICCV*, pages 1294–1303.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the CIFAR datasets. arXiv:1707.08819.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., and Yan, J. (2020). DARTS-: Robustly stepping out of performance collapse without indicators. arXiv:2009.01027.

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

Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., and Yang, S. (2017). Adanet: Adaptive structural learning of artificial neural networks. In *Proc. ICML*, pages 874–883.

Croce, F. and Hein, M. (2020). Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proc. ICML*, pages 2206–2216.

de G. Matthews, A. G., Hron, J., Rowland, M., Turner, R. E., and Ghahramani, Z. (2018). Gaussian process behaviour in wide deep neural networks. In *ICLR (Poster)*. OpenReview.net.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*.

Devries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15.

Dong, X. and Yang, Y. (2019a). One-shot neural architecture search via self-evaluated template network. In *Proc. ICCV*, pages 3680–3689.

Dong, X. and Yang, Y. (2019b). Searching for a robust neural architecture in four GPU hours. In *Proc. CVPR*, pages 1761–1770.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proc. ICLR*.

Du, S. S., Zhai, X., Póczos, B., and Singh, A. (2019). Gradient descent provably optimizes over-parameterized neural networks. In *Proc. ICLR*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019a). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:55:1–55:21.

Elsken, T., Metzen, J. H., and Hutter, F. (2019b). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21.

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. arXiv:1912.02757.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proc. ICML*, pages 1050–1059.

Ghadimi, S. and Lan, G. (2013). Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23:2341–2368.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *Proc. ICLR*.

Goodfellow, I. J. and Vinyals, O. (2015). Qualitatively characterizing neural network optimization problems. In *Proc. ICLR*.

Guo, M., Yang, Y., Xu, R., Liu, Z., and Lin, D. (2020). When NAS meets robustness: In search of robust architectures against adversarial attacks. In *Proc. CVPR*, pages 628–637.

Hardt, M., Recht, B., and Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. In *Proc. ICML*, pages 1225–1234.

He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. ICCV*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778.

Hong, W., Li, G., Zhang, W., Tang, R., Wang, Y., Li, Z., and Yu, Y. (2020). Dropnas: Grouped operation dropout for differentiable architecture search. In *Proc. IJCAI*, pages 2326–2332.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017a). Snapshot ensembles: Train 1, get M for free. In *Proc. ICLR*.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017b). Densely connected convolutional networks. In *Proc. CVPR*, pages 2261–2269.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer.

Jacot, A., Hongler, C., and Gabriel, F. (2018). Neural Tangent Kernel: Convergence and generalization in neural networks. In *Proc. NeurIPS*, pages 8580–8589.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with Gumbel-Softmax. In *Proc. ICLR*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proc. ICLR*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proc. ICLR*.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1106–1114.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proc. NIPS*, pages 6402–6413.

Larsson, G., Maire, M., and Shakhnarovich, G. (2017). Fractalnet: Ultra-deep neural networks without residuals. In *Proc. ICLR*.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. (2012). Efficient backprop. In *Neural Networks: Tricks of the Trade (2nd ed.)*, Lecture Notes in Computer Science, pages 9–48.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019a). Wide neural networks of any depth evolve as linear models under gradient descent. In *Proc. NeurIPS*, pages 8570–8581.

Lee, N., Ajanthan, T., and Torr, P. H. S. (2019b). Snip: Single-shot network pruning based on connection sensitivity. In *Proc. ICLR*.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Proc. NeurIPS*, pages 6391–6401.

Li, L. and Talwalkar, A. (2019a). Random search and reproducibility for neural architecture search. In *Proc. UAI*, page 129.

Li, L. and Talwalkar, A. (2019b). Random search and reproducibility for neural architecture search. In *Proc. UAI*, pages 367–377.

Lin, T., Maire, M., Belongie, S. J., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Proc. ECCV*, pages 740–755.

Liu, C., Dollár, P., He, K., Girshick, R. B., Yuille, A. L., and Xie, S. (2020). Are labels necessary for neural architecture search? In *Proc. ECCV*, pages 798–813.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. (2018). Progressive neural architecture search. In *Proc. ECCV*, pages 19–35.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *Proc. ICLR*.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Proc. NIPS*, pages 2370–2378.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T. (2018a). Neural architecture optimization. In *Proc. NeurIPS*, pages 7827–7838.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T. (2018b). Neural architecture optimization. In *Proc. NeurIPS*, pages 7827–7838.

Ma, N., Zhang, X., Zheng, H., and Sun, J. (2018). ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proc. ECCV*, pages 122–138.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, page 3.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The Concrete distribution: A continuous relaxation of discrete random variables. In *Proc. ICLR*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *Proc. ICLR*.

Mellor, J., Turner, J., Storkey, A. J., and Crowley, E. J. (2020a). Neural architecture search without training. arXiv:2006.04647.

Mellor, J., Turner, J., Storkey, A. J., and Crowley, E. J. (2020b). Neural architecture search without training. arXiv:2006.04647.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., and Zelnik-Manor, L. (2019). XNAS: Neural architecture search with expert advice. In *Proc. NeurIPS*, pages 1975–1985.

Nesterov, Y. (2004). *Introductory Lectures on Convex Optimization - A Basic Course*, volume 87 of *Applied Optimization*. Springer.

Parikh, N. and Boyd, S. P. (2014). Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239.

Park, D. S., Lee, J., Peng, D., Cao, Y., and Sohl-Dickstein, J. (2020). Towards NNGP-guided neural architecture search. arXiv:2011.06006.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *Proc. ICML*, pages 4092–4101.

Raghu, M., Poole, B., Kleinberg, J. M., Ganguli, S., and Sohl-Dickstein, J. (2017). On the expressive power of deep neural networks. In *Proc. ICML*, pages 2847–2854.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019a). Regularized evolution for image classifier architecture search. In *Proc. AAAI*, pages 4780–4789.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019b). Regularized

evolution for image classifier architecture search. In *Proc. AAAI*, pages 4780–4789.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the convergence of Adam and beyond. In *Proc. ICLR*.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. (2019). Evaluating the search phase of neural architecture search. Technical report.

Shu, Y., Cai, S., Dai, Z., Ooi, B. C., and Low, B. K. H. (2021). NASI: label- and data-agnostic neural architecture search at initialization. *CoRR*, abs/2109.00817.

Shu, Y., Wang, W., and Cai, S. (2020). Understanding architectures learnt by cell-based neural architecture search. In *Proc. ICLR*.

Sim, R. H. L., Zhang, Y., Low, B. K. H., and Jaillet, P. (2021). Collaborative bayesian optimization with fair regret. In *Proc. ICML*, pages 9691–9701.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Proc. NIPS*, pages 2960–2968.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Stahlberg, F. (2020). Neural machine translation: A review and survey.

Strauss, T., Hanselmann, M., Junginger, A., and Ulmer, H. (2017). Ensemble methods as a defense to adversarial perturbations against deep neural networks. arXiv:1709.03423.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015a). Going deeper with convolutions. In *Proc. CVPR*, pages 1–9.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015b). Going deeper with convolutions. In *Proc. CVPR*, pages 1–9.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019a). Mnasnet: Platform-aware neural architecture search for mobile. In *Proc. CVPR*, pages 2820–2828.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019b). MnasNet: Platform-aware neural architecture search for mobile. In *Proc. CVPR*, pages 2820–2828.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. In *Proc. NeurIPS*.

Turner, J., Crowley, E. J., O'Boyle, M. F. P., Storkey, A. J., and Gray, G. (2020). BlockSwap: Fisher-guided block substitution for network compression on a budget. In *Proc. ICLR*.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *J. Mach. Learn. Res.*, 9(11).

Wang, C., Zhang, G., and Grosse, R. B. (2020a). Picking winning tickets before training by preserving gradient flow. In *Proc. ICLR*.

Wang, C., Zhang, G., and Grosse, R. B. (2020b). Picking winning tickets before training by preserving gradient flow. In *Proc. ICLR*.

Xie, S., Kirillov, A., Girshick, R., and He, K. (2019a). Exploring randomly wired neural networks for image recognition. Technical report.

Xie, S., Zheng, H., Liu, C., and Lin, L. (2019b). SNAS: Stochastic neural architecture search. In *Proc. ICLR*.

Xu, J., Zhao, L., Lin, J., Gao, R., Sun, X., and Yang, H. (2021). KNAS: green neural architecture search. In *Proc. ICML*, pages 11613–11625.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., and Xiong, H. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *Proc. ICLR*.

Yang, G. and Littwin, E. (2021). Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In *Proc. ICML*, pages 11762–11772.

Yao, Q., Xu, J., Tu, W., and Zhu, Z. (2020). Efficient neural architecture search via proximal iterations. In *Proc. AAAI*, pages 6664–6671.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *Proc. ICML*, pages 7105–7114.

Zaidi, S., Zela, A., Elsken, T., Holmes, C., Hutter, F., and Teh, Y. W. (2020). Neural ensemble search for performant and calibrated predictions. arXiv:2006.08573.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020a). Understanding and robustifying differentiable architecture search. In *Proc. ICLR*.

Zela, A., Siems, J., and Hutter, F. (2020b). NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proc. ICLR*.

Zhao, Z., Zheng, P., Xu, S., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE Trans. Neural Networks Learn. Syst.*, 30:3212–3232.

Zhou, Z., Zhang, Q., and So, A. M.-C. (2015). 1,p-norm regularization: Error bounds and convergence rate analysis of first-order methods. In *Proc. ICML*, pages 1501–1510.

Zhou, Z.-H. (2012). *Ensemble methods: Foundations and algorithms*. CRC press.

Zhuo, J., Liu, C., Shi, J., Zhu, J., Chen, N., and Zhang, B. (2018). Message passing stein variational gradient descent. In *Proc. ICML*, pages 6013–6022.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *Proc. ICLR*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proc. CVPR*, pages 8697–8710.

Appendix A

Appendix for Chapter 4

A.1 Experimental Setup

A.1.1 Data Pre-processing and Augmentation

Our experiments are conducted on CIFAR-10/100 (Krizhevsky et al., 2009) and Tiny-ImageNet-200. CIFAR-10/100 contains 50,000 training images and 10,000 test images of 32×32 pixels in 10 and 100 classes respectively. Tiny-ImageNet-200 consists of 100,000 training images, 10,000 validation images and 10,000 test images[1] in 200 classes. We adopt the same data pre-processing and argumentation as described in DARTS (Liu et al., 2019): zero padding the training images with 4 pixels on each side and then randomly cropping them back to 32×32 on CIFAR-10/100 and 64×64 on Tiny-ImageNet-200; randomly flipping training images horizontally; normalizing training images with the means and standard deviations along the channel dimension.

[1] Since no label is attached to Tiny-ImageNet-200 test dataset, we instead use its validation dataset to get the generalization performance of various architectures. We still name it as the test accuracy/error for brief.

A.1.2 Sampling of Random Variants

For a N-node NAS cell, there are $\frac{(N-2)!}{(M-1)!}$ possible connections with M input nodes and one output node. There are therefore hundreds to thousands of possible randomly connected variants for each popular NAS cell. The random variants of operations consist of a similar or even higher amount of architectures. Due to the prohibitive cost of comparing popular NAS cells with all variants, we randomly sample some variants to understand why the popular NAS cells are selected.

Given a NAS cell C, we fix the partial order of intermediate nodes and their accompanying operations. We then replace the source node of their associated operations by uniformly randomly sampling a node from their proceeding nodes in the same cell to get their randomly connected variants. Similarly, given a NAS cell C, we fix the partial order of intermediate nodes and their connection topologies. We then replace the operations couping each connection by uniformly randomly sampling from candidate operations to get their random variants of operations.

A.1.3 Architectures and Training Details

For experiments on CIFAR-10/100 and Tiny-ImageNet-200, the neural network architectures are constructed by stacking $L = 20$ cells. Feature maps are down-sampled at the $L/3$-th and $2L/3$-th cell of the entire architecture with stride 2. For Tiny-ImageNet-200, the stride of the first convolutional layer is adapted to 2 to reduce the input resolution from 64×64 to 32×32. A more detailed building scheme can be found in DARTS (Liu et al., 2019).

In the default training setting, we apply stochastic gradient descent (SGD) with learning rate 0.025, momentum 0.9, weight decay 3×10^{-4}

and batch size 80 to train the models for 600 epochs on CIFAR10/100 and 300 epochs on Tiny-ImageNet-200 to ensure the convergence. The learning rate is gradually annealed to zero following the standard cosine annealing schedule. To compare the convergence under different learning rates in Section 4.3.1, we change the initial learning rate from 0.025 to 0.25 and 0.0025 respectively.

A.1.4 Regularization

Since regularization mechanisms shall affect the convergence (Zhou et al., 2015), architectures are trained without regularization for a neat empirical study in Section 4.3. The regularization mechanisms are only used in Section 4.4 to get the converged generalization performance of the original and adapted NAS architectures on CIFAR-10/100 and Tiny-ImageNet-200 as shown in Table 4.1.

There are three adopted regularization mechanisms on CIFAR-10/100 and Tiny-ImageNet-200 in this work: cutout (Devries and Taylor, 2017), auxiliary tower (Szegedy et al., 2015b) and drop path (Larsson et al., 2017). We apply standard cutout regularization with cutout length 16. Moreover, the auxiliary tower is located at $2L/3$-th cell of the entire architecture with weight 0.4. We apply the same linearly-increased drop path schedule as in NASNet (Zoph et al., 2018) with the maximum probability of 0.2.

A.2 Theoretical Analysis

A.2.1 Basics

We firstly compare the gradient of case I and case II shown in Figure 4.9. For case I, since $\boldsymbol{y}^{(i)} = W^{(i)}\boldsymbol{x}$, the gradient to each weight matrix $W^{(i)}$ is

denoted by

$$\frac{\partial f}{\partial W^{(i)}} = \frac{\partial f}{\partial \boldsymbol{y}^{(i)}} \boldsymbol{x}^T \tag{A.1}$$

Similarly, since $\widehat{\boldsymbol{y}}^{(i)} = \prod_{k=1}^{i} W^{(k)} \boldsymbol{x}$ for the case II, the gradient to each weight matrix $W^{(i)}$ is denoted by

$$\frac{\partial \widehat{f}}{\partial W^{(i)}} = \sum_{k=i}^{n} \Big(\prod_{j=i+1}^{k} W^{(j)} \Big)^T \frac{\partial \widehat{f}}{\partial \widehat{\boldsymbol{y}}^{(k)}} \Big(\prod_{j=1}^{i-1} W^{(j)} \boldsymbol{x} \Big)^T \tag{A.2}$$

$$= \sum_{k=i}^{n} \Big(\prod_{j=i+1}^{k} W^{(j)} \Big)^T \frac{\partial \widehat{f}}{\partial \widehat{\boldsymbol{y}}^{(k)}} \boldsymbol{x}^T \Big(\prod_{j=1}^{i-1} W^{(j)} \Big)^T \tag{A.3}$$

$$= \sum_{k=i}^{n} \Big(\prod_{j=i+1}^{k} W^{(j)} \Big)^T \frac{\partial f}{\partial W^{(k)}} \Big(\prod_{j=1}^{i-1} W^{(j)} \Big)^T \tag{A.4}$$

Exploring the fact that $\frac{\partial \widehat{f}}{\partial \widehat{\boldsymbol{y}}^{(i)}} = \frac{\partial f}{\partial \boldsymbol{y}^{(i)}}$, we get (4) by inserting (1) into (3).

A.2.2 Proof of Theorem 4.3

Due to the complexity of comparing the standard Lipschitz constant of the smoothness for these two cases, we instead investigate the block-wise Lipschitz constant (Beck and Tetruashvili, 2013). In other words, we evaluate the Lipschitz constant for each weight matrix $W^{(i)}$ while fixing all other matrices. Formally, we assume the block-wise Lipschitz smoothness of case I as

$$\left\| \frac{\partial f}{\partial W_1^{(i)}} - \frac{\partial f}{\partial W_2^{(i)}} \right\| \leq L^{(i)} \left\| W_1^{(i)} - W_2^{(i)} \right\| \quad \forall \; W_1^{(i)}, W_2^{(i)} \tag{A.5}$$

The default matrix norm we adopted is 2-norm. And $W_1^{(i)}, W_2^{(i)}$ denote

possible assignments for $W^{(i)}$.

Denoting that $\lambda^{(i)} = \left\|W^{(i)}\right\|$, which is the largest eigenvalue of matrix $W^{(i)}$, we can get the smoothness of case II as

$$\left\|\frac{\partial \widehat{f}}{\partial W_1^{(i)}} - \frac{\partial \widehat{f}}{\partial W_2^{(i)}}\right\| = \left\|\sum_{k=i}^{n}(\prod_{j=i+1}^{k} W^{(j)})^T(\frac{\partial f}{\partial W_1^{(k)}} - \frac{\partial f}{\partial W_2^{(k)}})(\prod_{j=1}^{i-1} W^{(j)})^T\right\| \tag{A.6}$$

$$\leq \sum_{k=i}^{n}\left\|(\prod_{j=i+1}^{k} W^{(j)})^T(\frac{\partial f}{\partial W_1^{(k)}} - \frac{\partial f}{\partial W_2^{(k)}})(\prod_{j=1}^{i-1} W^{(j)})^T\right\| \tag{A.7}$$

$$\leq \sum_{k=i}^{n}(\frac{1}{\lambda^{(i)}}\prod_{j=1}^{k}\lambda^{(j)})L^{(k)}\left\|W_1^{(k)} - W_2^{(k)}\right\| \tag{A.8}$$

$$\leq (\prod_{j=1}^{i-1}\lambda^{(j)})L^{(i)}\left\|W_1^{(i)} - W_2^{(i)}\right\| \tag{A.9}$$

We get the equality in (6) since $j > i$ and $W^{(j)}$ keeps the same for the computation of block-wise Lipschitz constant of $W^{(i)}$. Based on the triangle inequality of norm, we get (7) from (6). We get (8) from (7) based on the inequality $\|WV\| \leq \|W\|\|V\|$ and the assumption of the smoothness for case I in (5). Finally, since we are evaluating the block-wise Lipschitz constant for $W^{(i)}$, $W_1^{(k)} = W_2^{(k)}$ while $k \neq i$, which leads to the final inequality (9).

A.2.3 Proof of Theorem 4.4

Similarly, we assume the gradient variance of case I is bounded as

$$\mathbb{E}\left\|\frac{\partial f}{\partial W^{(i)}} - \mathbb{E}\frac{\partial f}{\partial W^{(i)}}\right\|^2 \leq (\sigma^{(i)})^2 \tag{A.10}$$

The gradient variance of case II is then bounded by

$$\mathbb{E}\left\|\frac{\partial \widehat{f}}{\partial W^{(i)}}-\mathbb{E}\frac{\partial \widehat{f}}{\partial W^{(i)}}\right\|^2=\mathbb{E}\left\|\sum_{k=i}^{n}(\prod_{j=i+1}^{k}W^{(j)})^T(\frac{\partial f}{\partial W^{(k)}}-\mathbb{E}\frac{\partial f}{\partial W^{(k)}})(\prod_{j=1}^{i-1}W^{(j)})^T\right\|^2 \tag{A.11}$$

$$\leq n\mathbb{E}\sum_{k=i}^{n}\left\|(\prod_{j=i+1}^{k}W^{(j)})^T(\frac{\partial f}{\partial W^{(k)}}-\mathbb{E}\frac{\partial f}{\partial W^{(k)}})(\prod_{j=1}^{i-1}W^{(j)})^T\right\|^2 \tag{A.12}$$

$$\leq n\sum_{k=i}^{n}(\frac{\sigma^{(k)}}{\lambda^{(i)}}\prod_{j=1}^{k}\lambda^{(j)})^2 \tag{A.13}$$

We get (12) from (11) based on Cauchy-Schwarz inequality. Based on the inequality $\|WV\| \leq \|W\|\|V\|$ and the assumption of bounded gradient variance for case I in (10), we get the final inequality.

Appendix B

Appendix for Chapter 5

B.1 Proofs

Proposition. (Training fairness in supernet.) *Let T and $T_{\mathcal{A}}$ denote the number of steps applied to train the supernet and candidate architecture $\mathcal{A}$ in the search space of size N, by uniformly randomly sampling a single architecture from this search space for the model training in each step, we have*

$$\Pr(\lim_{T\to\infty} T_{\mathcal{A}_i}/T = \lim_{T\to\infty} T_{\mathcal{A}_j}/T) = 1 \quad \forall i, j \in \{1, \cdots, N\}\,.$$

Proof. Let random variable $X_i^t \in \{0, 1\}$ denote the selection of candidate architecture $\mathcal{A}_i$ at training step t under our sampling scheme in the proposition above. For any $t > 0$ and $i, j \in [N]$, random variable $X_i^t - X_j^t$ can achieve following possible assignments and probabilities (denoted by p):

$$X_i^t - X_j^t = \begin{cases} +1, & p = 1/N \\ 0, & p = (N-2)/N \\ -1, & p = 1/N \end{cases}. \tag{B.1}$$

Consequently, $\mathbb{E}[X_i^t - X_j^t] = 0$. According to the strong law of large num-

bers, we further have

$$\Pr(\lim_{T\to\infty} T^{-1}\sum_{t=1}^{T} X_i^t - X_j^t = 0) = 1\ . \tag{B.2}$$

Note that

$$T^{-1}(T_{\mathcal{A}_i} - T_{\mathcal{A}_j}) = T^{-1}\sum_{t=1}^{T} X_i^t - X_j^t\ . \tag{B.3}$$

We thus can complete this proof by

$$\Pr(\lim_{T\to\infty} T^{-1}(T_{\mathcal{A}_i} - T_{\mathcal{A}_j}) = 0) = 1\ . \tag{B.4}$$

□

Proof of Proposition 5.1. As particles $\{\boldsymbol{x}_i\}_{i=1}^n$ of size n are applied to approximate the density q in our SVGD-RD, the second term (i.e., the controllable diversity term) in our (5.5) can then be approximated using these particles as

$$n\delta\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim q}\left[k(\boldsymbol{x},\boldsymbol{x}')\right] \approx \delta/n\sum_{i=1}^{n}\sum_{j=1}^{n} k(\boldsymbol{x}_i,\boldsymbol{x}_j) \triangleq \sum_{i=1}^{n} h(\boldsymbol{x}_i)\ , \tag{B.5}$$

where $h(\boldsymbol{x}) \triangleq \delta/n\sum_{j=1}^{n} k(\boldsymbol{x},\boldsymbol{x}_j)$. We take $\boldsymbol{x}_j$ in $k(\boldsymbol{x},\boldsymbol{x}_j)$ as a constant for the approximation above. Consequently, we have

$$\nabla_{\boldsymbol{x}_k}\sum_{i=1}^{n} h(\boldsymbol{x}_i) = \nabla_{\boldsymbol{x}_k} h(\boldsymbol{x}_k)\ . \tag{B.6}$$

Let $\boldsymbol{x}_i^+ \triangleq \boldsymbol{x}_i + \epsilon\boldsymbol{\phi}^*(\boldsymbol{x}_i)$ $(\forall i \in \{1,\cdots,n\})$ denote the functional gradient decent in the RKHS $\mathcal{H}$ to minimize the KL divergence term in our (5.5). Based on (B.6) above, given proximal operator $\mathrm{prox}_h(\boldsymbol{x}^+) = \arg\min_{\boldsymbol{y}} h(\boldsymbol{y}) + 1/2\|\boldsymbol{y}-\boldsymbol{x}^+\|_2^2$, by using proximal gradient method Parikh and Boyd (2014), our (5.5) can then be optimized via the following update to each particle

$\boldsymbol{x}_i$:

$$\boldsymbol{x}_i \leftarrow \text{prox}_h(\boldsymbol{x}_i^+) = \arg\min_{\boldsymbol{y}} h(\boldsymbol{y}) + 1/2\|\boldsymbol{y} - \boldsymbol{x}_i^+\|_2^2 \,. \tag{B.7}$$

According to the *Karush-Kuhn-Tucker* (KKT) conditions, the local optimum $\boldsymbol{y}^*$ of this proximal operator satisfies

$$\text{prox}_h(\boldsymbol{x}_i^+) = \boldsymbol{y}^* = \boldsymbol{x}_i^+ - \nabla_{\boldsymbol{y}^*} h(\boldsymbol{y}^*) \,. \tag{B.8}$$

When $h(\cdot)$ is convex, this local optimum is also a global optimum. As (16) is intractable to solve given a complex $h(\cdot)$, we approximate $h(\boldsymbol{y}^*)$ with its first-order Taylor expansion, i.e., $h(\boldsymbol{y}^*) \approx h(\boldsymbol{x}_i) + \nabla_{\boldsymbol{x}_i} h(\boldsymbol{x}_i)(\boldsymbol{y}^* - \boldsymbol{x}_i)$ and achieve following approximation:

$$\begin{aligned} \text{prox}_h(\boldsymbol{x}_i^+) &\approx \boldsymbol{x}_i^+ - \nabla_{\boldsymbol{x}_i} h(\boldsymbol{x}_) \\ &\approx \boldsymbol{x}_i + \epsilon \boldsymbol{\phi}^*(\boldsymbol{x}_i) - \nabla_{\boldsymbol{x}_i} h(\boldsymbol{x}_i) \\ &\approx \boldsymbol{x}_i + \epsilon \boldsymbol{\phi}^*(\boldsymbol{x}_i) - \delta/n \textstyle\sum_j^n \nabla_{\boldsymbol{x}_i} k(\boldsymbol{x}_i, \boldsymbol{x}_j) \,. \end{aligned} \tag{B.9}$$

Given the approximation $\boldsymbol{\phi}^*(\boldsymbol{x}_i) \approx \widehat{\boldsymbol{\phi}}^*(\boldsymbol{x}_i)$ and the definition of $\widehat{\boldsymbol{\phi}}^*(\boldsymbol{x}_i)$ in (2.8), we complete our proof by

$$\begin{aligned} \boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + 1/n \textstyle\sum_{j=1}^n k(\boldsymbol{x}_j, \boldsymbol{x}_i) \nabla_{\boldsymbol{x}_j} \log p(\boldsymbol{x}_j) \\ + \nabla_{\boldsymbol{x}_j} k(\boldsymbol{x}_j, \boldsymbol{x}_i) - \delta \nabla_{\boldsymbol{x}_i} k(\boldsymbol{x}_j, \boldsymbol{x}_i) \,. \end{aligned} \tag{B.10}$$

Proof of Proposition 5.2. Notably, since $k(\boldsymbol{x}, \boldsymbol{x}') = c$ when $\boldsymbol{x} = \boldsymbol{x}'$, we will achieve a constant $k(\boldsymbol{x}, \boldsymbol{x})$ for any particle $\boldsymbol{x}$ in the case of $n = 1$, which can be ignored in our SVGD-RD for any $\delta \in \mathbb{R}$. In light of this, our SVGD-RD in the case of $n = 1$ degenerates into standard SVGD. Consequently, to prove Proposition 5.2, we only need to consider SVGD in the case of $n = 1$.

Considering SVGD in the case of $n = 1$, we can frame the density q

represented by a single particle $\boldsymbol{x}'$ as

$$q(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} = \boldsymbol{x}' \\ 0 & \boldsymbol{x} \neq \boldsymbol{x}' \end{cases} . \tag{B.11}$$

The KL divergence between $q(\boldsymbol{x})$ and the target density $p(\boldsymbol{x})$ can then be simplified as

$$\mathrm{KL}(q\|p) = \mathbb{E}_{q(\boldsymbol{x})}[\log(q(\boldsymbol{x})/p(\boldsymbol{x}))] = -\log p(\boldsymbol{x}') . \tag{B.12}$$

Finally, standard SVGD in the case of $n = 1$ obtain its optimal particle by optimizing the following problem:

$$\begin{aligned} q^* &= \arg\min_q \mathrm{KL}(q\|p) \\ &= \arg\min_{\boldsymbol{x}'} \{-\log p(\boldsymbol{x}')\} \\ &= \arg\max_{\boldsymbol{x}'} p(\boldsymbol{x}') , \end{aligned} \tag{B.13}$$

which finally concludes the proof.

Remark. In practice, this $k(\boldsymbol{x}, \boldsymbol{x}) = c$ can be well satisfied, such as the radial basis function (RBF) kernel we applied in our experiments.

Appendix C

Appendix for Chapter 6

C.1 Theorems and Proofs

We firstly introduce the theorems proposed by Jacot et al. (2018); Lee et al. (2019a), which reveal the capability of NTK in characterizing the training dynamics of infinite-width DNNs. Note that the these theorems follow the parameterization and initialization of DNNs in (Jacot et al., 2018). Our analysis based on these theorems thus also needs to follow such parameterization and initialization of DNNs.

C.1.1 Neural Tangent Kernel at Initialization

Jacot et al. (2018) have validated that the outputs of infinite-width DNNs at initialization tends to Gaussian process (Theorem C.1) and have further revealed the deterministic limit of NTK at initialization (Theorem C.2). We denote $\mathbb{I}_{n_L}$ as a $n_L \times n_L$ matrix with all elements being 1.

Theorem C.1 (Jacot et al. (2018)). *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as $n_1, \cdots, n_{L-1} \to \infty$ sequentially, the output functions $f_{\theta,i}$ for $i = 1, \cdots, n_L$, tend (in law) to iid centered*

Gaussian processes of covariance $\Sigma^{(L)}$*, where* $\Sigma^{(L)}$ *is defined recursively by*

$$\Sigma^{(1)}(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{n_0}\boldsymbol{x}^\top\boldsymbol{x}' + \beta^2 \,,$$
$$\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x}') = \mathbb{E}_{g\sim\mathcal{N}(0,\Sigma^{(L-1)})}\Big[\sigma(g(\boldsymbol{x}))\sigma(g(\boldsymbol{x}'))\Big] + \beta^2$$

such that the expectation is with respect to a centered Gaussian process $\boldsymbol{f}$ *with covariance* $\Sigma^{(L-1)}$*.*

Theorem C.2 (Jacot et al. (2018))**.** *For a network of depth* L *at initialization, with a Lipschitz nonlinearity* σ*, and in the limit as* $n_1,\cdots,n_{L-1}\to\infty$ *sequentially, the NTK* $\boldsymbol{\Theta}^{(L)}$ *converges in probability to a deterministic limiting kernel:*

$$\boldsymbol{\Theta}^{(L)} \to \boldsymbol{\Theta}_\infty^{(L)} \otimes \mathbb{I}_{n_L} \,.$$

Kernel $\boldsymbol{\Theta}_\infty^{(L)} : \mathbb{R}^{n_0\times n_0}\to\mathbb{R}$ *is defined recursively by*

$$\boldsymbol{\Theta}_\infty^{(1)}(\boldsymbol{x},\boldsymbol{x}') = \Sigma^{(1)}(\boldsymbol{x},\boldsymbol{x}') \,,$$
$$\boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{\Theta}_\infty^{(L-1)}(\boldsymbol{x},\boldsymbol{x}')\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x}') + \Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x}') \,,$$

where

$$\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x}') = \mathbb{E}_{g\sim\mathcal{N}(0,\Sigma^{(L-1)})}\Big[\dot{\sigma}(g(\boldsymbol{x}))\dot{\sigma}(g(\boldsymbol{x}'))\Big]$$

such that the expectation is with respect to a centered Gaussian process $\boldsymbol{f}$ *with covariance* $\Sigma^{(L-1)}$ *and* $\dot{\sigma}$ *denotes the derivative of* σ*.*

C.1.2 Training Dynamics of Infinite-Width Neural Networks

Given $\lambda_{\min}(\mathbf{\Theta})$ as the minimal eigenvalue of NTK $\mathbf{\Theta}$ and define $\eta_{\text{critical}} \triangleq 2(\lambda_{\min}(\mathbf{\Theta}) + \lambda_{\max}(\mathbf{\Theta}))^{-1}$, Lee et al. (2019a) have characterized the training dynamics of infinite-wide neural networks as below.

Theorem C.3 (Lee et al. (2019a)). *Let $n_1 = \cdots = n_{L-1} = k$ and assume $\lambda_{\min}(\mathbf{\Theta}) > 0$. Applying gradient descent with learning rate $\eta < \eta_{\text{critical}}$ (or gradient flow), for every $\boldsymbol{x} \in \mathbb{R}^{n_0}$ with $\|\boldsymbol{x}\| \leq 1$, with probability arbitrarily close to 1 over random initialization,*

$$\sup_{t \geq 0} \left\| f_t - f_t^{\text{lin}} \right\|_2 = \mathcal{O}(k^{-\frac{1}{2}}) \quad \text{as } k \to \infty \,.$$

Remark. For the case of $L = 2$, Du et al. (2019) have revealed that if any two input vectors of a dataset are not parallel, then $\lambda_{\min}(\mathbf{\Theta}) > 0$ holds, which fortunately can be well-satisfied for most real-world datasets. Though the training dynamics of DNNs only tend to be governed by their linearization at initialization when the infinite width is satisfied as revealed in Theorem C.3, empirical results in (Lee et al., 2019a) suggest that such linearization can also govern the training dynamics of practical over-parameterized DNNs.

C.1.3 Proof of Proposition 6.1

With aforementioned theorems, especially Theorem C.3, our Proposition 6.1 can be proved as below with an introduced lemma (Lemma C.1).

Lemma C.1. *Let loss function $\mathcal{L}\big(f(\mathcal{X};\theta_t),\mathcal{Y}\big)$, abbreviated to $\mathcal{L}(f_t)$, be γ-Lipschitz continuous within the domain $\mathcal{V}$. Under the condition in Theo-*

rem C.3, there exists a constant $c_0 > 0$ such that as $k \to \infty$,

$$\left\|\mathcal{L}(\boldsymbol{f}_t) - \mathcal{L}(\boldsymbol{f}_t^{\text{lin}})\right\|_2 \leq \frac{c_0 \gamma}{\sqrt{k}}$$

with probability arbitrarily close to 1.

Proof. Since $\mathcal{L}(\boldsymbol{f}_t)$ is γ-Lipschitz continuous, for any $\boldsymbol{v}, \boldsymbol{u} \in \mathcal{V}$, we have

$$\left\|\mathcal{L}(\boldsymbol{v}) - \mathcal{L}(\boldsymbol{u})\right\|_2 \leq \gamma \left\|\boldsymbol{v} - \boldsymbol{u}\right\|_2 \tag{C.1}$$

following the definition of Lipschitz continuity. Besides, under the condition in Theorem C.3, Theorem C.3 reveals that there exists a constant c_0 such that as $k \to \infty$,

$$\left\|\boldsymbol{f}_t - \boldsymbol{f}_t^{\text{lin}}\right\|_2 \leq \frac{c_0}{\sqrt{k}} \tag{C.2}$$

with probability arbitrarily close to 1. Combining (C.1) and (C.2), we hence can finish the proof by

$$\left\|\mathcal{L}(\boldsymbol{f}_t) - \mathcal{L}(\boldsymbol{f}_t^{\text{lin}})\right\|_2 \leq \gamma \left\|\boldsymbol{f}_t - \boldsymbol{f}_t^{\text{lin}}\right\|_2 \leq \frac{c_0 \gamma}{\sqrt{k}} \quad \text{as } k \to \infty\,. \tag{C.3}$$

□

Remark. The γ-Lipschitz continuity based on $\|\cdot\|_2$ is commonly satisfied for widely adopted loss functions. For example, given 1-dimensional MSE $\mathcal{L} = m^{-1}\sum_i^m (x_i - y_i)^2$, let $x_i, y_i \in [0,1]$ denote the prediction and label respectively, the Lipschitz of MSE with respect to $\boldsymbol{x} \triangleq (x_1, \cdots, x_m)^\top$ is 2. Meanwhile, given the n-class Cross Entropy with Softmax $\mathcal{L} = -m^{-1}\sum_i^m \sum_j^n y_{i,j} \log(p_{i,j})$ as the loss function, let $p_{i,j} \in (0,1)$ and $y_{i,j} \in \{0,1\}$ denote the prediction and label correspondingly, with $\sum_j y_{i,j} = 1$ and $p_{i,j} \triangleq \exp(x_{i,j}) / \sum_j \exp(x_{i,j})$ for input $x_{i,j} \in \mathbb{R}$, the Lipschitz of Cross Entropy with Softmax with respect to $\boldsymbol{x} \triangleq (x_{1,1}, \cdots, x_{i,j}, \cdots, x_{m,n})^\top$ is then

1.

Proof of Proposition 6.1. Note that the linearization $f^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}_t)$ achieves a constant NTK $\boldsymbol{\Theta}_t = \boldsymbol{\Theta}_0$ because its gradient with respect to $\boldsymbol{\theta}_t$ (i.e., $\nabla_{\boldsymbol{\theta}_0} f(\boldsymbol{x};\boldsymbol{\theta}_0)$) stays constant over time. Given MSE as the loss function, according to the loss decomposition (2.12) in Sec. 2.4, the training dynamics of $f^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}_t)$ can then be analyzed in a closed form:

$$\mathcal{L}(f_t^{\text{lin}}) = \frac{1}{m}\sum_{i=1}^{mn}(1-\eta\lambda_i)^{2t}(\boldsymbol{u}_i^\top \mathcal{Y})^2 \,, \tag{C.4}$$

where $\boldsymbol{\Theta}_0 = \sum_{i=1}^{mn}\lambda_i(\boldsymbol{\Theta}_0)\boldsymbol{u}_i\boldsymbol{u}_i^\top$, and $\lambda_i(\boldsymbol{\Theta}_0)$ and $\boldsymbol{u}_i$ denote the i-th largest eigenvalue and the corresponding eigenvector of $\boldsymbol{\Theta}_0$, respectively. With $\eta < \lambda_{\max}(\boldsymbol{\Theta}_0)^{-1}$ and $\lambda_{\min}(\boldsymbol{\Theta}_0) > 0$, $\eta\lambda_i(\boldsymbol{\Theta}_0)$ is then under the constraint that

$$0 < \eta\lambda_i(\boldsymbol{\Theta}_0) < \frac{\lambda_{\max}(\boldsymbol{\Theta}_0)}{\lambda_{\max}(\boldsymbol{\Theta}_0)} \Rightarrow 0 < \eta\lambda_i(\boldsymbol{\Theta}_0) < 1 \,. \tag{C.5}$$

Hence, for the case of $t \geq 0.5$, with $0 < 1-\eta\lambda_i(\boldsymbol{\Theta}_0) < 1$ and $\overline{\lambda}(\boldsymbol{\Theta}_0) \triangleq (mn)^{-1}\sum_{i=1}^{mn}\lambda_i(\boldsymbol{\Theta}_0)$,

$$\begin{aligned}\sum_{i=1}^{mn}(1-\eta\lambda_i(\boldsymbol{\Theta}_0))^{2t} &\leq \sum_{i=1}^{mn}(1-\eta\lambda_i(\boldsymbol{\Theta}_0)) \\ &= mn(1-\eta\overline{\lambda}(\boldsymbol{\Theta}_0)) \,.\end{aligned} \tag{C.6}$$

Further, given $0 \leq t < 0.5$, the scalar function $y = x^{2t}$ is concave for any $x \in \mathbb{R}_{\geq 0}$. Following from the Jensen's inequality on this concave function,

$$\begin{aligned}\sum_{i=1}^{mn}(1-\eta\lambda_i(\boldsymbol{\Theta}_0))^{2t} &\leq mn\left[\frac{1}{mn}\sum_{i=1}^{mn}(1-\eta\lambda_i(\boldsymbol{\Theta}_0))\right]^{2t} \\ &= mn(1-\eta\overline{\lambda}(\boldsymbol{\Theta}_0))^{2t} \,.\end{aligned} \tag{C.7}$$

With bounded labels $\mathcal{Y} \in [0,1]^{mn}$ and the unit norm of eigenvectors $\|\boldsymbol{u}_i\|_2 = 1$,

$$(\boldsymbol{u}_i^\top \mathcal{Y})^2 \leq \|\boldsymbol{u}_i\|_2^2 \|\mathcal{Y}\|_2^2 \leq \|\mathcal{Y}\|_2^2 \leq mn \,. \tag{C.8}$$

By introducing (C.6), (C.7) and (C.8) into (C.4), the training loss $\mathcal{L}(\boldsymbol{f}_t^{\text{lin}})$ can then be bounded by

$$\begin{aligned} \mathcal{L}(\boldsymbol{f}_t^{\text{lin}}) &\leq (mn) \cdot \frac{1}{m} \sum_{i=1}^{mn} (1 - \lambda_i(\boldsymbol{\Theta}_0))^{2t} \\ &\leq mn^2 (1 - \eta \overline{\lambda}(\boldsymbol{\Theta}_0))^q \,, \end{aligned} \tag{C.9}$$

where q is set to be $2t$ if $0 \leq t < 0.5$, and 1 otherwise.

Following from Theorem C.3, while applying gradient descent (or gradient flow) with learning rate $\eta < \lambda_{\max}^{-1} < \eta_{\text{critical}} \triangleq 2(\lambda_{\min} + \lambda_{\max})^{-1}$, for all $\boldsymbol{x} \in \mathcal{X}$ with $\|\boldsymbol{x}\| \leq 1$, there exists a constant c_0 such that as $k \to \infty$,

$$\left\| \boldsymbol{f}_t - \boldsymbol{f}_t^{\text{lin}} \right\|_2 \leq \frac{c_0}{\sqrt{k}} \tag{C.10}$$

with probability arbitrarily close to 1. Hence, $\boldsymbol{f}_t^{\text{lin}} \in \left[-c_0\sqrt{1/k}, 1 + c_0\sqrt{1/k}\right]^{mn}$ with $\boldsymbol{f}_t \in [0,1]^{mn}$. Within the extended domain $\boldsymbol{f}_t \in \left[-c_0\sqrt{1/k}, 1 + c_0\sqrt{1/k}\right]^{mn}$ for the MSE loss function,

$$\begin{aligned} \left\| \nabla_{\boldsymbol{f}_t} \mathcal{L} \right\|_2 &= \frac{2}{m} \left\| \mathcal{Y} - \boldsymbol{f}_t \right\|_2 \\ &\leq 2\sqrt{n/m} \left(1 + c_0\sqrt{1/k}\right) \,. \end{aligned} \tag{C.11}$$

Hence, the MSE within this extended domain is $2\sqrt{n/m}\left(1 + c_0\sqrt{1/k}\right)$-Lipschitz continuous.

Combining with Lemma C.1,

$$\begin{aligned} \mathcal{L}(\boldsymbol{f}_t) &\leq \mathcal{L}(\boldsymbol{f}_t^{\text{lin}}) + 2c_0\sqrt{n/(mk)}\left(1 + c_0\sqrt{1/k}\right) \\ &\leq mn^2(1 - \eta\overline{\lambda})^q + 2c_0\sqrt{n/(mk)}\left(1 + c_0\sqrt{1/k}\right) \,, \end{aligned} \tag{C.12}$$

which concludes the proof by denoting $\mathcal{L}_t \triangleq \mathcal{L}(f_t)$.

Remark. $\|\boldsymbol{x}\|_2 \leq 1$ can be well-satisfied for the normalized dataset, which is conventionally adopted as data pre-processing in practice.

C.1.4 Proof of Proposition 6.2

According to Theorem C.2, $\boldsymbol{\Theta}_0 \triangleq \boldsymbol{\Theta}^{(L)}$ is determined by both $\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x})$ and $\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x})$ of all $\boldsymbol{x} \in \mathcal{X}$. We hence introduce Lemma C.2 below regarding $\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x})$ and $\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x})$ to ease the proof of Proposition 6.2. Particularly, we set $\beta = 0$ for the β in Theorem C.1 and Theorem C.2 throughout our analysis. Note that this condition is usually satisfied by training DNNs without bias.

Lemma C.2. *For a network of depth L at initialization, with the γ-Lipschitz continuous nonlinearity σ satisfying $|\sigma(x)| \leq |x|$ for all $x \in \mathbb{R}$, given the input features $\boldsymbol{x} \in \mathcal{X}$,*

$$\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x}) \leq n_0^{-1}\boldsymbol{x}^\top\boldsymbol{x}\,, \quad \dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x}) \leq \gamma^2\,,$$

where $\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x})$ and $\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x})$ are defined in Theorem C.1 and Theorem C.2, respectively, and $\beta = 0$.

Proof. Following from the definition in Theorem C.1,

$$\begin{aligned}\Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x}) &= \mathbb{E}_{g\sim\mathcal{N}(0,\Sigma^{(L-1)})}\left[\sigma(g(\boldsymbol{x}))\sigma(g(\boldsymbol{x}))\right] \\ &\overset{(a)}{\leq} \mathbb{E}_{g\sim\mathcal{N}(0,\Sigma^{(L-1)})}\left[g^2(\boldsymbol{x})\right] \\ &\overset{(b)}{=} \Sigma^{(L-1)}(\boldsymbol{x},\boldsymbol{x})\,,\end{aligned} \tag{C.13}$$

in which (a) follows from $|\sigma(x)| \leq |x|$ and (b) results from the variance of

$g \sim \mathcal{N}(0, \Sigma^{(L-1)})$. By following (C.13) from layer L to 1, we get

$$\Sigma^{(L)}(\boldsymbol{x}, \boldsymbol{x}) \leq \Sigma^{(1)}(\boldsymbol{x}, \boldsymbol{x}) \overset{(a)}{=} n_0^{-1} \boldsymbol{x}^\top \boldsymbol{x} \,, \tag{C.14}$$

where (a) is defined in Theorem C.1.

Similarly, following from the definition in Theorem C.2,

$$\begin{aligned} \dot{\Sigma}^{(L)}(\boldsymbol{x}, \boldsymbol{x}) &= \mathbb{E}_{g \sim \mathcal{N}(0, \Sigma^{(L-1)})} \left[\dot{\sigma}(g(\boldsymbol{x})) \dot{\sigma}(g(\boldsymbol{x})) \right] \\ &\overset{(a)}{\leq} \mathbb{E}_{g \sim \mathcal{N}(0, \Sigma^{(L-1)})} \left[\gamma^2 \right] \\ &\overset{(b)}{=} \gamma^2 \,, \end{aligned} \tag{C.15}$$

in which (a) results from the γ-Lipschitz continuity of nonlinearity and (b) follows from the expectation of a constant. □

Proof of Proposition 6.2. Notably, Theorem C.2 reveals that in the limit as $n_1, \cdots, n_{L-1} \to \infty$ sequentially, $\boldsymbol{\Theta}^{(L)} \to \boldsymbol{\Theta}_\infty^{(L)} \otimes \mathbb{I}_{n_L}$ with probability arbitrarily close to 1 over random initializations. We therefore only need to focus on this deterministic limiting kernel to simplify our analysis, i.e., let $\boldsymbol{\Theta}_0 = \boldsymbol{\Theta}_\infty^{(L)} \otimes \mathbb{I}_{n_L}$. Particularly, given m input vectors $\boldsymbol{x} \sim P(\boldsymbol{x})$ with covariance matrix Σ_P and a L-layer neural architecture of n-dimensional

output, For $\gamma \neq 1$, we have

$$
\begin{aligned}
(mn)^{-1}\|\boldsymbol{\Theta}_0\|_{\mathrm{tr}} &= (mn)^{-1}\|\boldsymbol{\Theta}_\infty^{(L)} \otimes \mathbb{I}_{n_L}\|_{\mathrm{tr}} \\
&\overset{(a)}{=} (mn)^{-1}\|\boldsymbol{\Theta}_\infty^{(L)}\|_{\mathrm{tr}}\|\mathbb{I}_{n_L}\|_{\mathrm{tr}} \\
&\overset{(b)}{=} m^{-1}\|\boldsymbol{\Theta}_\infty^{(L)}\|_{\mathrm{tr}} \\
&\overset{(c)}{=} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x})\right] \\
&\overset{(d)}{=} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{\Theta}_\infty^{(L-1)}(\boldsymbol{x},\boldsymbol{x})\dot{\Sigma}^{(L)}(\boldsymbol{x},\boldsymbol{x}) + \Sigma^{(L)}(\boldsymbol{x},\boldsymbol{x})\right] \\
&\overset{(e)}{\leq} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\gamma^2\boldsymbol{\Theta}_\infty^{(L-1)}(\boldsymbol{x},\boldsymbol{x}) + n_0^{-1}\boldsymbol{x}^\top\boldsymbol{x}\right] \\
&\overset{(f)}{=} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\gamma^2\boldsymbol{\Theta}_\infty^{(L-1)}(\boldsymbol{x},\boldsymbol{x})\right] + n_0^{-1}\|\Sigma_P\|_{\mathrm{tr}}\,,
\end{aligned}
\tag{C.16}
$$

in which (a) derives from the property of Kronecker product, (b) follows from the notation $n_L = n$ and (c) results from the property of expectation and trace norm. In addition, (d) follows from the definition of $\boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x})$ in Theorem C.2 and (e) is derived by introducing Lemma C.2 into (d). Finally, (f) follows from the expectation and variance of $P(\boldsymbol{x})$. By following (c-f) from layer L to 1, we can get

$$
\begin{aligned}
(mn)^{-1}\|\boldsymbol{\Theta}_0\|_{\mathrm{tr}} \leq\ & \gamma^{2(L-1)}\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\Sigma^{(1)}(\boldsymbol{x},\boldsymbol{x})\right] + \\
& n_0^{-1}\|\Sigma_P\|_{\mathrm{tr}}\left(1-\gamma^{2(L-1)}\right)\left(1-\gamma^2\right)^{-1} \\
=\ & n_0^{-1}\|\Sigma_P\|_{\mathrm{tr}}\left(1-\gamma^{2L}\right)\left(1-\gamma^2\right)^{-1}\,.
\end{aligned}
\tag{C.17}
$$

Notably, (C.17) is derived under the condition that $\gamma \neq 1$. For the case of $\gamma = 1$, similarly, we can get

$$
(mn)^{-1}\|\boldsymbol{\Theta}_0\|_{\mathrm{tr}} \leq n_0^{-1}L\|\Sigma_P\|_{\mathrm{tr}}\,. \tag{C.18}
$$

Note that with $\|\boldsymbol{x}\|_2 \leq 1$, we have

$$\begin{aligned} \|\Sigma_P\|_{\mathrm{tr}} &= \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{x}^\top \boldsymbol{x}\right] - \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{x}\right]^\top \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{x}\right] \\ &\leq \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{x}^\top \boldsymbol{x}\right] \\ &\leq 1\,. \end{aligned} \tag{C.19}$$

Given any two different distributions $P(\boldsymbol{x})$ and $Q(\boldsymbol{x})$, define $Z = \int \|P(\boldsymbol{x}) - Q(\boldsymbol{x})\| \mathrm{d}\boldsymbol{x}$,[1] we can construct a special probability density function $\widetilde{P}(\boldsymbol{x}) = Z^{-1}\|P(\boldsymbol{x}) - Q(\boldsymbol{x})\|$ to further employee the bounds in (C.17) and (C.18). Specifically, for $\gamma \neq 1$, by combining (C.17) and (C.19), we can get

$$\begin{aligned} (mn)^{-1}\left|\left\|\boldsymbol{\Theta}_0(P)\right\|_{\mathrm{tr}} - \left\|\boldsymbol{\Theta}_0(Q)\right\|_{\mathrm{tr}}\right| &\overset{(a)}{=} \left|\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[\boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim Q(\boldsymbol{x})}\left[\boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x})\right]\right| \\ &\overset{(b)}{\leq} \int \|P(\boldsymbol{x}) - Q(\boldsymbol{x})\| \boldsymbol{\Theta}_\infty^{(L)}(\boldsymbol{x},\boldsymbol{x})\, d\boldsymbol{x} \\ &\overset{(c)}{\leq} n_0^{-1} Z \|\Sigma_P\|_{\mathrm{tr}} \left(1-\gamma^{2L}\right)\left(1-\gamma^2\right)^{-1} \\ &\overset{(d)}{\leq} n_0^{-1} Z \left(1-\gamma^{2L}\right)\left(1-\gamma^2\right)^{-1}\,, \end{aligned} \tag{C.20}$$

in which (a) follows from (c) in (C.16), (b) results from Cauchy Schwarz inequality and (c) is obtained by introducing inequality (C.17) and distribution $\widetilde{P}(\boldsymbol{x})$. Finally, (d) is based on (C.19).

Similarly, in the case of $\gamma = 1$, we can get

$$(mn)^{-1}\left|\left\|\boldsymbol{\Theta}_0(P)\right\|_{\mathrm{tr}} - \left\|\boldsymbol{\Theta}_0(Q)\right\|_{\mathrm{tr}}\right| \leq n_0^{-1} Z L\,, \tag{C.21}$$

which concludes the proof.

Remark. Note that ReLU is widely adopted as the nonlinearity in neural networks, which satisfies the Lipschitz continuity with $\gamma = 1$ and the

[1] We abuse this integration to ease notations.

inequality $|\sigma(x)| \leq |x|$. Notably, many other ReLU-type nonlinearities (e.g., Leaky ReLU (Maas et al., 2013) and PReLU (He et al., 2015b)) can also satisfy these two conditions.

Appendix D

Appendix for Chapter 7

D.1 Proofs

Throughout the proofs in this work, we use lower-case bold-faced symbols to denote column vectors (e.g., $\boldsymbol{x}$), and upper-case bold-faced symbols to represent matrices (e.g., $\mathbf{A}$).

D.1.1 Proof of Theorem 7.1

D.1.1.0.1 Connecting $\mathcal{M}_{\text{Grad}}$ with $\mathcal{M}_{\text{Trace}}$. Since we have assumed that the loss function $\ell(\cdot,\cdot)$ is β-Lipschitz continuous in the first argument, we

have the following inequalities using the notations in Sec. 7.2:

$$\begin{aligned}
\mathcal{M}_{\text{Grad}} &= \left\| \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right\|_2 \\
&\leq \frac{1}{m} \sum_{i=1}^{m} \left\| \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right\|_2 \\
&\leq \frac{1}{m} \sum_{i=1}^{m} \left| \nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right| \|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\|_2 \\
&\leq \frac{\beta}{m} \sum_{i=1}^{m} \|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\|_2 \\
&\leq \frac{\beta}{m} \sqrt{m \sum_{i=1}^{m} \|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\|_2^2} \\
&= \beta \mathcal{M}_{\text{Trace}}
\end{aligned} \tag{D.1}$$

where we let $\nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i)$ be the gradient with respect to the output of DNN model denoted by f. Note that the first inequality derives from the Cauchy-Schwarz inequality. The third inequality results from the definition of Lipschitz continuity and the fourth inequality also follows from the Cauchy-Schwarz inequality. Finally, the last equality is based on the definition of NTK matrix defined in Sec. 7.2.1, i.e.,

$$\mathcal{M}_{\text{Trace}} = \sqrt{\frac{1}{m} \|\boldsymbol{\Theta}_0\|_{\text{tr}}} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\|_2^2} \,. \tag{D.2}$$

Let $C_1 = \beta$, with a high probability, we have

$$\mathcal{M}_{\text{Grad}} \leq C_1 \mathcal{M}_{\text{Trace}} \,. \tag{D.3}$$

D.1.1.0.2 Connecting $\mathcal{M}_{\text{SNIP}}$ with $\mathcal{M}_{\text{Grad}}$. We firstly introduce the following lemma.

Lemma D.1. *If* $\mathrm{x}_1, \cdots, \mathrm{x}_k$ *are independent standard normal random variables,*

for $\mathrm{y} = \sum_{i=1}^{k} \mathrm{x}_i^2$ *and any* ϵ,

$$\mathrm{P}(\mathrm{y} - k \geq 2\sqrt{k\epsilon} + 2\epsilon) \leq \exp(-\epsilon) \,.$$

Following that of (Jacot et al., 2018; Arora et al., 2019a), each element of $\boldsymbol{\theta}_0 \in \mathbb{R}^d$ follows from the standard normal distribution independently. We therefore can bound $\|\boldsymbol{\theta}_0\|_2^2$ using the lemma above. Specifically, let $\delta = \exp(-\epsilon) \in (0,1)$, with probability at least $1-\delta$ over random initialization, we have:

$$\|\boldsymbol{\theta}_0\|_2^2 \leq d + 2\sqrt{d \log \frac{1}{\delta}} + 2 \log \frac{1}{\delta} \,. \tag{D.4}$$

Using the results above and following the definition of $\mathcal{M}_{\text{Grad}}$, with probability at least $1-\delta$ over random initialization, we have

$$\begin{aligned}
\mathcal{M}_{\text{SNIP}} &= \frac{1}{m} \sum_{i=1}^{m} \left| \boldsymbol{\theta}_0^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right| \\
&\leq \frac{1}{m} \sum_{i=1}^{m} \|\boldsymbol{\theta}_0\|_2 \|\nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i)\|_2 \\
&\leq \sqrt{d + 2\sqrt{d \log \frac{1}{\delta}} + 2 \log \frac{1}{\delta}} \cdot \frac{1}{m} \sum_{i=1}^{m} \left\| \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \right\|_2 \\
&\leq \beta \sqrt{d + 2\sqrt{d \log \frac{1}{\delta}} + 2 \log \frac{1}{\delta}} \mathcal{M}_{\text{Trace}}
\end{aligned} \tag{D.5}$$

where the last inequality follows the same derivation in (D.1). Let $C_2 = \beta \sqrt{d + 2\sqrt{d \log \frac{1}{\delta}} + 2 \log \frac{1}{\delta}}$, with a high probability, we finally have

$$\mathcal{M}_{\text{SNIP}} \leq C_2 \mathcal{M}_{\text{Trace}} \,. \tag{D.6}$$

D.1.1.0.3 Connecting $\mathcal{M}_{\text{GraSP}}$ and $\mathcal{M}_{\text{Grad}}$. We firstly introduce the following lemma adapted from (Lee et al., 2019a).

Lemma D.2 (Lemma 1 in (Lee et al., 2019a))**.** *Let* $\delta \in (0,1)$. *There exist*

constant $\rho_1, \rho_2 > 0$ *such that for any* $r > 0$, $\boldsymbol{\theta}, \boldsymbol{\theta}' \in B(\boldsymbol{\theta}_0, r/\sqrt{n})$ *and any input* $\boldsymbol{x}$ *within the dataset, with probability at least* $1-\delta$ *over random initialization, we have*

$$\left\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})\right\|_2 \leq \rho_1$$

$$\left\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}'} f(\boldsymbol{x}, \boldsymbol{\theta}')\right\|_2 \leq \rho_2 \left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|_2$$

where $B(\boldsymbol{\theta}_0, r/\sqrt{n}) \triangleq \{\boldsymbol{\theta} : \|\boldsymbol{\theta} - \boldsymbol{\theta}_0\| \leq r/\sqrt{n}\}$.

To ease the notation, we use $\nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i)$ to denote the gradient with respect to the output of DNN (i.e., $f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)$). Following the definition of Hessian matrix, $\mathbf{H}_i$ applied in $\mathcal{M}_{\text{GraSP}}$ can be computed as

$$\begin{aligned}
\mathbf{H}_i &= \nabla^2_{\boldsymbol{\theta}_0} \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \\
&= \nabla_{\boldsymbol{\theta}} \left[\nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right] \\
&= \nabla^2_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top + \nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \,.
\end{aligned} \tag{D.7}$$

Since $\ell(\cdot, \cdot)$ is assumed to be γ-Lipschitz smooth and β-Lipschitz continuous in the first argument, we can then bound the operator norm of this hessian matrix $\mathbf{H}_i$ based on the input $\boldsymbol{x}_i$ in the dataset by

$$\begin{aligned}
\|\mathbf{H}_i\|_2 &= \left\|\nabla^2_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top + \nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i) \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2 \\
&\leq \left|\nabla^2_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i)\right| \left\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top\right\|_2 + \left|\nabla_f \ell(f(\boldsymbol{x}_i, \boldsymbol{\theta}_0), y_i)\right| \left\|\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2 \\
&\leq \gamma \left\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top\right\|_2 + \beta \left\|\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2 \\
&\leq \gamma \operatorname{Tr}(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top) + \beta \left\|\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2 \\
&\leq \gamma \left\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2^2 + \beta \left\|\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)\right\|_2 \\
&\leq \gamma \rho_1^2 + \beta \rho_2
\end{aligned} \tag{D.8}$$

where the last inequality results from Lemma D.2 and is satisfied with probability at least $1-\delta$ over random initialization.

Finally, based on the definition of $\mathcal{M}_{\text{GraSP}}$, with probability at least

$1-2\delta$ over random initialization, we have.

$$\begin{aligned}
\mathcal{M}_{\text{GraSP}} &= \frac{1}{m}\left|\sum_{i=1}^{m}\boldsymbol{\theta}_0^{\top}(\mathbf{H}_i\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{x}_i,\boldsymbol{\theta}_0),y_i))\right| \\
&\leq \frac{1}{m}\|\boldsymbol{\theta}_0\|_2\sum_{i=1}^{m}\left\|\mathbf{H}_i\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{x}_i,\boldsymbol{\theta}_0),y_i)\right\|_2 \\
&\leq \frac{1}{m}\|\boldsymbol{\theta}_0\|_2\sum_{i=1}^{m}\|\mathbf{H}_i\|_2\|\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\boldsymbol{x}_i,\boldsymbol{\theta}_0),y_i)\|_2 \\
&\leq (\gamma\rho_1^2+\beta\rho_2)\sqrt{d+2\sqrt{d\log\frac{2}{\delta}}+2\log\frac{2}{\delta}}\cdot\frac{1}{m}\sum_{i=1}^{m}\left\|\nabla_{\boldsymbol{\theta}}\ell(f(\boldsymbol{x}_i,\boldsymbol{\theta}_0),y_i)\right\|_2 \\
&\leq \beta(\gamma\rho_1^2+\beta\rho_2)\sqrt{d+2\sqrt{d\log\frac{2}{\delta}}+2\log\frac{2}{\delta}}\mathcal{M}_{\text{Trace}}\,.
\end{aligned} \tag{D.9}$$

Similarly, let $C_3=\beta(\gamma\rho_1^2+\beta\rho_2)\sqrt{d+2\sqrt{d\log\frac{2}{\delta}}+2\log\frac{2}{\delta}}$, with a high probability, we finally have

$$\mathcal{M}_{\text{GraSP}}\leq C_3\mathcal{M}_{\text{Trace}}\,, \tag{D.10}$$

which concludes our proof.

Remark. Let the training-free metric applied in (Xu et al., 2021) be defined as

$$\mathcal{M}_{\text{KNAS}}\triangleq\sqrt{\left|\frac{1}{m^2}\sum_{i,j=1}^{m}\nabla_{\boldsymbol{\theta}}f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)^{\top}\nabla_{\boldsymbol{\theta}}f(\boldsymbol{x}_j,\boldsymbol{\theta}_0)\right|}\,. \tag{D.11}$$

Interestingly, $\mathcal{M}_{\text{KNAS}}$ is also gradient-based according to Sec. 7.2.2. As a result, we can also theoretically connect $\mathcal{M}_{\text{KNAS}}$ with $\mathcal{M}_{\text{Trace}}$ in a similar

way:

$$\begin{aligned}\mathcal{M}_{\text{KNAS}}^2 &= \left| \frac{1}{m^2} \sum_{i,j=1}^{m} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_j, \boldsymbol{\theta}_0) \right| \\ &\leq \frac{1}{m^2} \sqrt{m^2 \sum_{i,j=1}^{m} \left(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_j, \boldsymbol{\theta}_0) \right)^2} \\ &= \frac{1}{m} \|\boldsymbol{\Theta}_0\|_{\text{F}} \leq \frac{1}{m} \|\boldsymbol{\Theta}_0\|_{\text{tr}} = \mathcal{M}_{\text{Trace}}^2\end{aligned} \tag{D.12}$$

where the first inequality follows from the Cauchy-Schwarz inequality and the second equality is based on the definition of Frobenius norm. The last inequality derives from the matrix inequality $\|\cdot\|_{\text{F}} \leq \|\cdot\|_{\text{tr}}$ while the last equality is obtained based on the definition of $\mathcal{M}_{\text{Trace}}$. Therefore, we have the following theoretical connection between $\mathcal{M}_{\text{KNAS}}$ and $\mathcal{M}_{\text{Trace}}$, which we validate empirically in Sec. 7.6.1.

$$\mathcal{M}_{\text{KNAS}} \leq \mathcal{M}_{\text{Trace}} \,. \tag{D.13}$$

Remark. Note that our assumptions about the Lipschitz continuity and the Lipschitz smoothness of loss function $\ell(\cdot, \cdot)$ can be well-satisfied by the commonly employed loss functions in practice, e.g., Cross Entropy and Mean Square Error. For example, Shu et al. (2021) have justified that these two commonly applied loss functions indeed satisfy the Lipschitz continuous assumption. As for their Lipschitz smoothness, following a similar analysis in (Shu et al., 2021), we can also easily verify that there exists a constant $c > 0$ such that $\|\nabla_f^2 \ell(f, \cdot)\|_2 \leq c$ for both Cross Entropy and Mean Square Error.

D.1.2 Proof of Theorem 7.2

D.1.2.1 Estimating the Rademacher Complexity of DNNs

We firstly the Rademacher complexity of a hypothesis $\mathcal{G}$ over dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ of size m as

$$\mathcal{R}_S(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \epsilon_i g(\boldsymbol{x}_i)\right], \tag{D.14}$$

with $\epsilon_i \in \{\pm 1\}$. Let $\boldsymbol{\theta}_0$ be the initialized parameters of DNN model f, we then define the following hypothesises that will be used in our lemmas and theorems:

$$\mathcal{F} \triangleq \{\boldsymbol{x} \mapsto f(\boldsymbol{x}, \boldsymbol{\theta}_t) : t > 0\}, \quad \mathcal{F}^{\text{lin}} \triangleq \{\boldsymbol{x} \mapsto f(\boldsymbol{x}, \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}_0} f(\boldsymbol{x}, \boldsymbol{\theta}_0)^\top (\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) : t > 0\}, \tag{D.15}$$

where $f_t \in \mathcal{F}$ and $f_t^{\text{lin}} \in \mathcal{F}^{\text{lin}}$ denote the functions determined by the DNN model f and its corresponding linearization at step t of their model training, respectively. Note that the $\boldsymbol{\theta}_t$ in f_t and f_t^{lin} are not the same and are determined by the optimization of f_t and f_t^{lin} correspondingly. Interestingly, f_t can then be well characterized by f_t^{lin} as proved in the following lemma.

Lemma D.3 (Theorem H.1 (Lee et al., 2019a)). *Let $n_1 = \cdots = n_{L-1} = n$ and assume $\lambda_{\min}(\boldsymbol{\Theta}_\infty) > 0$. There exist the constant $c > 0$ and $N > 0$ such that for any $n > N$ and any $\boldsymbol{x} \in \mathbb{R}^{n_0}$ with $\|\boldsymbol{x}\|_2 \leq 1$, the following holds with probability at least $1 - \delta$ over random initialization when applying gradient descent with learning rate $\eta < \eta_0$,*

$$\sup_{t\geq 0} \left\| f_t - f_t^{\text{lin}} \right\|_2 \leq \frac{c}{\sqrt{n}}.$$

Remark. As revealed in (Lee et al., 2019a), $\lambda_{\min}(\boldsymbol{\Theta}_\infty) > 0$ indeed holds

especially when any input $\boldsymbol{x}$ from dataset S satisfies $\|\boldsymbol{x}\|_2 = 1$. In practice, $\|\boldsymbol{x}\|_2 = 1$ can be achieved by normalizing each input $\boldsymbol{x}$ from real-world dataset using its norm $\|\boldsymbol{x}\|_2$.

Moreover, we will show that the Rademacher complexity of DNN model during model training (i.e., $\mathcal{F}$) can also be bounded using its linearization model (i.e., $\mathcal{F}^{\text{lin}}$) based on the following lemmas.

Lemma D.4. *Under the conditions in Lemma D.3, there exists a constant $c > 0$ such that with probability at least $1 - \delta$ over random initialization, the following holds*

$$\mathcal{R}_S(\mathcal{F}) \leq \mathcal{R}_S(\mathcal{F}^{lin}) + \frac{c}{\sqrt{n}} .$$

Proof. Based on Lemma D.3, given $\epsilon_i \in \{\pm 1\}$, with probability at least $1 - \delta$, there exists a constant $c > 0$ such that

$$\epsilon_i f_t \leq \epsilon_i f_t^{\text{lin}} + \frac{c}{\sqrt{n}} . \tag{D.16}$$

Following the definition of Rademacher complexity, we can bound the complexity of $\mathcal{F}$ by

$$\begin{aligned}
\mathcal{R}_S(\mathcal{F}) &= \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{f\in\mathcal{F}} \frac{1}{m}\sum_{i=1}^{m} \epsilon_i f(\boldsymbol{x}_i, \boldsymbol{\theta})\right] \\
&\leq \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{f^{\text{lin}}\in\mathcal{F}^{\text{lin}}} \frac{1}{m}\sum_{i=1}^{m} \left(\epsilon_i f^{\text{lin}}(\boldsymbol{x}_i) + \frac{c}{\sqrt{n}}\right)\right] \\
&\leq \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{f^{\text{lin}}\in\mathcal{F}^{\text{lin}}} \frac{1}{m}\sum_{i=1}^{m} \epsilon_i f^{\text{lin}}(\boldsymbol{x}_i)\right] + \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\frac{c}{\sqrt{n}}\right] \\
&\leq \mathcal{R}_S(\mathcal{F}^{\text{lin}}) + \frac{c}{\sqrt{n}} ,
\end{aligned} \tag{D.17}$$

which completes the proof. □

Lemma D.5. *Let $f(\mathbf{X}, \boldsymbol{\theta}_0) \triangleq [f(\boldsymbol{x}_1, \boldsymbol{\theta}_0) \cdots f(\boldsymbol{x}_m, \boldsymbol{\theta}_0)]^\top$ and $\boldsymbol{y} \triangleq [y_1 \cdots y_m]^\top$ be*

the predictions of DNN model f at initialization and the target labels of a dataset $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, respectively. Define empirical loss $\mathcal{L} \triangleq \sum_{i=1}^m \|f^{\text{lin}}(\boldsymbol{x}_i, \boldsymbol{\theta}) - y_i\|_2^2/(2m)$ and NTK matrix at initialization $\boldsymbol{\Theta}_0 \triangleq \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)^\top$, assume $\lambda_{\min}(\boldsymbol{\Theta}_0) > 0$, for any $t > 0$, the following holds when applying gradient descent on $f^{\text{lin}}(\boldsymbol{x}, \boldsymbol{\theta})$ with learning rate $\eta < m/\lambda_{\max}(\boldsymbol{\Theta}_0)$

$$\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_0\|_2 \leq \|\boldsymbol{\theta}_\infty - \boldsymbol{\theta}_0\|_2 = \sqrt{\widehat{\boldsymbol{y}}^\top \boldsymbol{\Theta}_0^{-1} \widehat{\boldsymbol{y}}},$$

where $\boldsymbol{\theta}_t$ denotes the parameters of f^{lin} at step t of its model training and $\widehat{\boldsymbol{y}} \triangleq \boldsymbol{y} - f(\mathbf{X}, \boldsymbol{\theta}_0)$. Besides, $\lambda_{\max}(\boldsymbol{\Theta}_0)$ and $\lambda_{\min}(\boldsymbol{\Theta}_0)$ denote the maximal and minimal eigenvalue of matrix $\boldsymbol{\Theta}_0$.

Proof. Following the update of gradient descent on MSE with learning rate $\eta < m/\lambda_{\max}(\boldsymbol{\Theta}_0)$, we have

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{m} \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)^\top \left(f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_t) - \boldsymbol{y} \right). \tag{D.18}$$

By subtracting $\boldsymbol{\theta}_0$, multiplying $\nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)$ and adding $f(\mathbf{X}, \boldsymbol{\theta}_0)$ on both sides of the equation above, we achieve

$$f(\mathbf{X}, \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_0) = f(\mathbf{X}, \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) - \frac{\eta}{m} \boldsymbol{\Theta}_0 \left(f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_t) - \boldsymbol{y} \right), \tag{D.19}$$

which can be simplified as

$$\begin{aligned} f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_{t+1}) &= f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_t) - \frac{\eta}{m} \boldsymbol{\Theta}_0 \left[f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_t) - \boldsymbol{y} \right] \\ &= \left(\mathbf{I} - \frac{\eta}{m} \boldsymbol{\Theta}_0 \right) f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_t) + \frac{\eta}{m} \boldsymbol{\Theta}_0 \boldsymbol{y}. \end{aligned} \tag{D.20}$$

By recursively applying the equality above, we finally achieve

$$\begin{aligned} f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_{t+1}) &= \left(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{t+1} f^{\text{lin}}(\mathbf{X}, \boldsymbol{\theta}_0) + \sum_{j=0}^{t}\left(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{j}\left(\frac{\eta}{m}\boldsymbol{\Theta}_0 \boldsymbol{y}\right) \\ &= \left(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{t+1} f(\mathbf{X}, \boldsymbol{\theta}_0) + \left[\mathbf{I} - (\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0)^{t+1}\right]\left(\frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{-1}\frac{\eta}{m}\boldsymbol{\Theta}_0 \boldsymbol{y} \\ &= \left(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{t+1}\left(f(\mathbf{X}, \boldsymbol{\theta}_0) - \boldsymbol{y}\right) + \boldsymbol{y}\,, \end{aligned} \tag{D.21}$$

where the last equality derives from the sum of matrix series with $\eta < m/\lambda_{\max}(\boldsymbol{\Theta}_0)$. Note that this result can be integrated into (D.18) and provide the following explicit form of $\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_0$ after applying gradient descent for $t+1$ times:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_0 &= \sum_{k=0}^{t} \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k \\ &= \frac{\eta}{m}\nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)^\top \sum_{k=0}^{t}\left(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0\right)^{k}\left(\boldsymbol{y} - f(\mathbf{X}, \boldsymbol{\theta}_0)\right) \\ &= \frac{\eta}{m}\nabla_{\boldsymbol{\theta}_0} f(\mathbf{X}, \boldsymbol{\theta}_0)^\top \sum_{k=0}^{t}(\mathbf{I} - \frac{\eta}{m}\boldsymbol{\Theta}_0)^{k}\widehat{\boldsymbol{y}} \end{aligned} \tag{D.22}$$

Since $\boldsymbol{\Theta}_0$ is symmetric, we can also represent $\boldsymbol{\Theta}_0$ as $\boldsymbol{\Theta}_0 = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$ using principal component analysis (PCA), where $\mathbf{V}$ and $\boldsymbol{\Lambda}$ denotes the matrix of eigenvectors $\{\boldsymbol{v}_i\}_{i=1}^m$ and eigenvalues $\{\lambda_i\}_{i=1}^m$, respectively. Based on this

representation, we have

$$
\begin{aligned}
\|\boldsymbol{\theta}_{t+1}-\boldsymbol{\theta}_0\|_2 &= \frac{\eta}{m}\sqrt{(\boldsymbol{\theta}_{t+1}-\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}_{t+1}-\boldsymbol{\theta}_0)} \\
&= \frac{\eta}{m}\sqrt{\widehat{\boldsymbol{y}}^\top \sum_{k=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Theta}_0)^k \nabla_{\boldsymbol{\theta}_0} f(\mathbf{X},\boldsymbol{\theta}_0)\nabla_{\boldsymbol{\theta}_0} f(\mathbf{X},\boldsymbol{\theta}_0)^\top \sum_{k'=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Theta}_0)^{k'}\widehat{\boldsymbol{y}}} \\
&= \frac{\eta}{m}\sqrt{\widehat{\boldsymbol{y}}^\top \sum_{k=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Theta}_0)^k \boldsymbol{\Theta}_0 \sum_{k'=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Theta}_0)^{k'}\widehat{\boldsymbol{y}}} \\
&= \frac{\eta}{m}\sqrt{\widehat{\boldsymbol{y}}^\top \sum_{k=0}^{t}(\mathbf{I}-\frac{\eta}{m}\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top)^k \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top \sum_{k'=0}^{t}(\mathbf{I}-\frac{\eta}{m}\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top)^{k'}\widehat{\boldsymbol{y}}} \\
&= \frac{\eta}{m}\sqrt{\widehat{\boldsymbol{y}}^\top \mathbf{V} \sum_{k=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Lambda})^k \mathbf{V}^\top\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top\mathbf{V} \sum_{k'=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Lambda})^{k'}\mathbf{V}^\top\widehat{\boldsymbol{y}}} \\
&= \frac{\eta}{m}\sqrt{\widehat{\boldsymbol{y}}^\top \mathbf{V} \sum_{k=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Lambda})^k \boldsymbol{\Lambda} \sum_{k'=0}^{t}(\mathbf{I}-\frac{\eta}{m}\boldsymbol{\Lambda})^{k'}\mathbf{V}^\top\widehat{\boldsymbol{y}}} \\
&= \frac{\eta}{m}\sqrt{\sum_{i=1}^{m}\lambda_i\left[\sum_{k=0}^{t}(1-\frac{\eta}{m}\lambda_i)^k\right]^2(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2}\,.
\end{aligned}
\tag{D.23}
$$

Since $\eta < m/\lambda_{\max}(\boldsymbol{\Theta}_0)$ and $\lambda_{\min}(\boldsymbol{\Theta}_0) > 0$, we have $0 < 1-\eta\lambda_i/m < 1$ and hence

$$
\begin{aligned}
\|\boldsymbol{\theta}_t-\boldsymbol{\theta}_0\|_2 &= \frac{\eta}{m}\sqrt{\sum_{i=1}^{m}\lambda_i\left[\sum_{k=0}^{t-1}(1-\frac{\eta}{m}\lambda_i)^k\right]^2(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2} \\
&\le \frac{\eta}{m}\sqrt{\sum_{i=1}^{m}\lambda_i\left[\sum_{k=0}^{t}(1-\frac{\eta}{m}\lambda_i)^k\right]^2(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2} \\
&= \|\boldsymbol{\theta}_{t+1}-\boldsymbol{\theta}_0\|_2
\end{aligned}
\tag{D.24}
$$

We complete the proof by recursively applying the inequalities above

$$\begin{aligned}
\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_0\|_2 &\leq \|\boldsymbol{\theta}_\infty - \boldsymbol{\theta}_0\|_2 \\
&= \frac{\eta}{m}\sqrt{\sum_{i=1}^{m}\lambda_i\left[\sum_{k=0}^{\infty}(1-\frac{\eta}{m}\lambda_i)^k\right]^2(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2} \\
&= \frac{\eta}{m}\sqrt{\sum_{i=1}^{m}\lambda_i\left[\frac{1}{\eta\lambda_i/m}\right]^2(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2} \\
&= \sqrt{\sum_{i=1}^{m}\lambda_i^{-1}(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2} \\
&= \sqrt{\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}}}
\end{aligned} \tag{D.25}$$

□

Lemma D.6 (Awasthi et al. (2020)). *Let $\mathcal{G} \triangleq \{\boldsymbol{x} \mapsto \boldsymbol{w}^T\boldsymbol{x} : \|\boldsymbol{w}\|_2 \leq R\}$ be a family of linear functions defined over $\mathbb{R}^d$ with bounded weight. Then the empirical Rademacher complexity of $\mathcal{G}$ for m samples $S \triangleq (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m)$ admits the following upper bounds:*

$$\mathcal{R}_S(\mathcal{G}) \leq \frac{R}{m}\|\mathbf{X}^\top\|_{2,2}\,,$$

where $\mathbf{X}$ is the $d \times m$-matrix with $\boldsymbol{x}_i$s as columns: $\mathbf{X} \triangleq [\boldsymbol{x}_1 \cdots \boldsymbol{x}_m]$.

Based on our Lemma D.4 and Lemma D.5, we can finally bound the Rademacher complexity of a DNN model during its model training (i.e., $\mathcal{F}$) using its linearization model (i.e., $\mathcal{F}^{\text{lin}}$). Specifically, under the conditions in Theorem D.3 and Lemma D.5, there exist the constant $c > 0$ and $N > 0$ such that for any $n > N$, with probability at least $1 - \delta$ over

initialization, we have

$$\begin{aligned}
\mathcal{R}_S(\mathcal{F}) &\leq \mathcal{R}_S(\mathcal{F}^{\text{lin}}) + \frac{c}{\sqrt{n}} \\
&= \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{t\geq 0}\frac{1}{m}\sum_{i=1}^{m}\epsilon_i\Big(f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)+\nabla_{\boldsymbol{\theta}_0}f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}_t-\boldsymbol{\theta}_0)\Big)\right]+\frac{c}{\sqrt{n}} \\
&= \mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}\left[\sup_{t\geq 0}\frac{1}{m}\sum_{i=1}^{m}\epsilon_i\nabla_{\boldsymbol{\theta}_0}f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta}_t-\boldsymbol{\theta}_0)\right]+\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{\boldsymbol{\epsilon}\in\{\pm1\}^m}[\epsilon_i]f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)+\frac{c}{\sqrt{n}} \\
&\leq \frac{\|\boldsymbol{\theta}_\infty-\boldsymbol{\theta}_0\|_2\|\nabla_{\boldsymbol{\theta}_0}f(\mathbf{X},\boldsymbol{\theta}_0)\|_{2,2}}{m}+\frac{c}{\sqrt{n}} \\
&\leq \frac{\|\nabla_{\boldsymbol{\theta}_0}f(\mathbf{X},\boldsymbol{\theta}_0)\|_{2,2}}{m}\sqrt{\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}}}+\frac{c}{\sqrt{n}} \\
&\leq \sqrt{\kappa\lambda_0}\cdot\sqrt{\frac{\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}}}{m}}+\frac{c}{\sqrt{n}}\,,
\end{aligned} \tag{D.26}$$

where the first inequality derives from Lemma D.6 and the last inequality derives from the following inequalities based on the definition $\kappa \triangleq \lambda_{\max}(\boldsymbol{\Theta}_0)/\lambda_{\min}(\boldsymbol{\Theta}_0)$ and $\lambda_0 \triangleq \lambda_{\min}(\boldsymbol{\Theta}_0)$.

$$\begin{aligned}
\|\nabla_{\boldsymbol{\theta}_0}f(\mathbf{X},\boldsymbol{\theta}_0)\|_{2,2} &= \sqrt{\sum_{i}^{m}\|\nabla_{\boldsymbol{\theta}_0}f(\boldsymbol{x}_i,\boldsymbol{\theta}_0)\|_2^2} \\
&= \sqrt{\sum_{i=1}^{m}\lambda_i(\boldsymbol{\Theta}_0)} \\
&\leq \sqrt{m\kappa\lambda_0}\,.
\end{aligned} \tag{D.27}$$

D.1.2.2 Deriving the Generalization Bound for Training-free Metrics

Define the generalization error on data distribution $\mathcal{D}$ as $\mathcal{L}_{\mathcal{D}}(g) \triangleq \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\ell(g(\boldsymbol{x}),y)$ and the empirical error on dataset $S = \{(\boldsymbol{x}_i,y_i)\}_{i=1}^m$ randomly sampled from $\mathcal{D}$ as $\mathcal{L}_S(g) \triangleq \sum_{i=1}^m \ell(g(\boldsymbol{x}_i),y_i)$. Given the loss function $\ell(\cdot,\cdot)$ and the Rademacher complexity of any hypothesis $\mathcal{G}$, the generalization error on hypothesis $\mathcal{G}$ can then be estimated by the empirical error using the following lemma.

Lemma D.7 (Mohri et al. (2012)). *Suppose the loss function $\ell(\cdot,\cdot)$ is bounded in $[0,1]$ and is β-Lipschitz continuous in the first argument. Then with probability at least $1-\delta$ over dataset S of size m:*

$$\sup_{g\in\mathcal{G}}\{\mathcal{L}_{\mathcal{D}}(g)-\mathcal{L}_S(g)\}\leq 2\beta\mathcal{R}_S(\mathcal{G})+3\sqrt{\log(2/\delta)/(2m)}\,.$$

Lemma D.8. *For symmetric matrix $\mathbf{A}\in\mathbb{R}^{m\times m}$ with eigenvalues $\{\lambda_i\}_{i=1}^m$ in an ascending order, define $\kappa\triangleq\lambda_m/\lambda_1$, the following inequality holds if $\lambda_1>0$,*

$$\|\mathbf{A}\|_{\mathrm{tr}}\left\|\mathbf{A}^{-1}\right\|_{\mathrm{tr}}\leq m^2\kappa\,.$$

Proof. Since eigenvalues $\{\lambda_i\}_{i=1}^m$ are in an ascending order, we have

$$\frac{\lambda_m}{\kappa}\leq\lambda_i\leq\lambda_1\kappa\,. \tag{D.28}$$

Based on the results above, we can connect the matrix norm $\|\mathbf{A}\|_{\mathrm{tr}}$ and $\|\mathbf{A}^{-1}\|_{\mathrm{tr}}$ as below:

$$\|\mathbf{A}\|_{\mathrm{tr}}\left\|\mathbf{A}^{-1}\right\|_{\mathrm{tr}}=(\sum_{i=1}^m\lambda_i)\cdot(\sum_{i=1}^m\lambda_i^{-1})\leq(m\lambda_1\kappa)\cdot\frac{m\kappa}{\lambda_m}=\frac{m^2\kappa}{\kappa}=m^2\kappa\,, \tag{D.29}$$

which concludes the proof. □

Now we are prepared to prove our Theorem 7.2 by combining the results in Lemma D.7 and (D.26). Specifically, under the conditions in Theorem D.3 and Lemma D.5, there exist constant $c,N>0$ such that for any $f_t\in\mathcal{F}$ and any $n>N$, with probability at least $1-2\delta$ over random

initialization, we have

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(f_t) &\leq \mathcal{L}_S(f_t) + 2\beta\mathcal{R}_S(\mathcal{F}) + 3\sqrt{\frac{\log(2/\delta)}{2m}} \\ &\leq \mathcal{L}_S(f_t) + 2\beta\sqrt{\kappa\lambda_0}\cdot\sqrt{\frac{\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}}}{m}} + \frac{2\beta c}{\sqrt{n}} + 3\sqrt{\frac{\log(2/\delta)}{2m}}\,.\end{aligned} \tag{D.30}$$

Assume $f(\boldsymbol{x},\boldsymbol{\theta}_0)$ and y is bounded in $[0,1]$ for any pair $(\boldsymbol{x},y)$ in the dataset S, let $\{\boldsymbol{v}_i\}_{i=1}^m$ and $\{\lambda_i\}_{i=1}^m$ be the eigenvectors and eigenvalues of $\boldsymbol{\Theta}_0$, respectively, we have $\widehat{\boldsymbol{y}}\in[-1,1]^m$ and the following inequalities:

$$\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}} = \sum_i^m \frac{(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}})^2}{\lambda_i} \leq \sum_i^m \frac{\|\boldsymbol{v}_i\|_2^2\|\widehat{\boldsymbol{y}}\|_2^2}{\lambda_i} \leq \sum_i^m \frac{m}{\lambda_i}\,. \tag{D.31}$$

Based on the fact that $\|\boldsymbol{\Theta}_0\|_{\text{tr}} = \sum_{i=1}^m \lambda_i$ and Lemma D.8, we finally achieve

$$\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}} \leq m\left\|\boldsymbol{\Theta}_0^{-1}\right\|_{\text{tr}} \leq \frac{m^3\kappa}{\|\boldsymbol{\Theta}_0\|_{\text{tr}}} = \frac{m^2\kappa}{\mathcal{M}_{\text{Trace}}^2}\,. \tag{D.32}$$

By introducing (D.32) into (D.30), with $\lambda_0 \leq 1$, we have

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(f_t) &\leq \mathcal{L}_S(f_t) + 2\beta\sqrt{\kappa\lambda_0}\cdot\sqrt{\frac{\widehat{\boldsymbol{y}}^\top\boldsymbol{\Theta}_0^{-1}\widehat{\boldsymbol{y}}}{m}} + \frac{2\beta c}{\sqrt{n}} + 3\sqrt{\frac{\log(2/\delta)}{2m}} \\ &\leq \mathcal{L}_S(f_t) + \frac{2\beta\kappa\sqrt{m}}{\mathcal{M}_{\text{Trace}}} + \frac{2\beta c}{\sqrt{n}} + 3\sqrt{\frac{\log(2/\delta)}{2m}}\,.\end{aligned} \tag{D.33}$$

Let $\mathcal{M}$ be any metric introduced in Sec. 7.2.2, based on the results in our Theorem 7.1 and the definition of $\mathcal{O}(\cdot)$, the following inequality holds with high probability using the result above,

$$\mathcal{L}_{\mathcal{D}}(f_t) \leq \mathcal{L}_S(f_t) + \mathcal{O}(\kappa/\mathcal{M})\,, \tag{D.34}$$

which concludes our proof of Theorem 7.2.

Remark. The underlying assumption in our Theorem 7.2 is that $\lambda_{\min}(\boldsymbol{\Theta}_\infty) >$

0, which has been justified in the remark following Lemma D.3. As for $\lambda_{\min}(\mathbf{\Theta}_0) > 0$, it can be well-satisfied by introducing zero-mean noise into the gradient of model parameters and our Theorem 7.2 will still hold with high probability in this case. Though our conclusion is based on the initialization using standard normal distribution and over-parameterized DNNs, our empirical results in Sec. 7.6.5 show that this conclusion can also work well when DNNs is initialized using other methods or DNN width is relatively small.

D.1.3 Proof of Corollary 7.2

To prove our Corollary 7.2, we firstly consider the convergence of f_t^{lin} under the same conditions in Theorem 7.2. Specifically, following the notations and results in Lemma D.5, let $\{\boldsymbol{v}_i\}_{i=1}^m$ and $\{\lambda_i\}_{i=1}^m$ be the eigenvectors and eigenvalues of $\mathbf{\Theta}_0$, respectively, we have

$$\begin{aligned}
\mathcal{L}_S(f_t^{\text{lin}}) &= \frac{1}{2m}\left\|f^{\text{lin}}(\mathbf{X},\boldsymbol{\theta}_t)-\boldsymbol{y}\right\|_2^2 \\
&= \frac{1}{2m}\left\|\left(\mathbf{I}-\frac{\eta}{m}\mathbf{\Theta}_0\right)^t\left(f(\mathbf{X},\boldsymbol{\theta}_0)-\boldsymbol{y}\right)\right\|_2^2 \\
&= \frac{1}{2m}\left\|\left(\mathbf{I}-\frac{\eta}{m}\mathbf{\Theta}_0\right)^t\widehat{\boldsymbol{y}}\right\|_2 \\
&= \frac{1}{2m}\sum_{i=1}^m\left(1-\frac{\eta}{m}\lambda_i\right)^{2t}\left(\boldsymbol{v}_i^\top\widehat{\boldsymbol{y}}\right)^2 \\
&\leq \frac{1}{2m}\sum_{i=1}^m\left(1-\frac{\eta}{m}\lambda_i\right)^{2t}\|\boldsymbol{v}_i\|_2^2\left\|\widehat{\boldsymbol{y}}\right\|_2^2,
\end{aligned} \tag{D.35}$$

where the fourth equality follows the same derivation in (D.23). Moreover, under the assumption that $\widehat{\boldsymbol{y}}\in[0,1]^m$ and the fact that $\|\boldsymbol{v}_i\|_2 = 1$, for any

$t > 0$ (i.e., $t = 1, 2, \cdots$), we have

$$\begin{aligned}
\mathcal{L}_S(f_t^{\text{lin}}) &\leq \frac{1}{2}\sum_{i=1}^{m}\left(1-\frac{\eta}{m}\lambda_i\right)^{2t} \\
&\leq \frac{1}{2}m\left(\frac{1}{m}\sum_{i=1}^{m}1-\frac{\eta}{m}\lambda_i\right)^{2t} \\
&= \frac{1}{2}m\left(1-\frac{\eta}{m^2}\|\boldsymbol{\Theta}_0\|_{\text{tr}}\right)^{2t} \\
&= \frac{1}{2}m\left(1-\frac{\eta}{m}\mathcal{M}_{\text{Trace}}^2\right)^{2t} \\
&\leq \frac{1}{2}m\left(1-\frac{\eta}{mc}\mathcal{M}^2\right)^{2t}
\end{aligned} \tag{D.36}$$

where the second inequality derives from the fact that $0 < 1 - \eta\lambda_i/m < 1$ as $\eta < m/\lambda_{\max}(\boldsymbol{\Theta}_0)$ and $\lambda_{\min}(\boldsymbol{\Theta}_0) > 0$ and the Jensen's inequality. Besides, the last inequality is based on the results in our Theorem 7.1: For any training-free metric $\mathcal{M}$ introduced in Sec. 7.2.2, there exists a constants c such that the following holds with high probability,

$$\mathcal{M}^2 \leq c\mathcal{M}_{\text{Trace}}^2 \quad \Rightarrow \quad 1-\frac{\eta}{mc}\mathcal{M}^2 \geq 1-\frac{\eta}{m}\mathcal{M}_{\text{Trace}}^2\,. \tag{D.37}$$

Based on Lemma D.3 and the fact that loss function $\ell(f, y) = (f-y)^2/2$ is 1-Lipschitz continuous in the first argument, with high probability, we have

$$\left|\mathcal{L}_S(f_t) - \mathcal{L}_S(f_t^{\text{lin}})\right| \leq \left|f_t - f_t^{\text{lin}}\right| \leq \mathcal{O}(\frac{1}{\sqrt{n}})\,. \tag{D.38}$$

By introducing the results above into our Theorem 7.2, we finally achieve the following results with high probability,

$$\begin{aligned}
\mathcal{L}_{\mathcal{D}}(f_t) &\leq \mathcal{L}_S(f_t) + \mathcal{O}(\kappa/\mathcal{M}) \leq \mathcal{L}_S(f_t^{\text{lin}}) + \mathcal{O}(\kappa/\mathcal{M}) \\
&\leq \frac{1}{2}m\left(1-\frac{\eta}{mc}\mathcal{M}^2\right)^{2t} + \mathcal{O}(\kappa/\mathcal{M})\,,
\end{aligned} \tag{D.39}$$

which thus concludes our proof.

D.1.4 Proof of Theorem 7.3

We firstly introduce the following lemma, which is adapted from (Shu et al., 2021).

Lemma D.9 (Shu et al., 2021). *Suppose $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{x}\|_2 \leq 1$ for all $\boldsymbol{x}$ in a dataset of size m, a given L-layer DNN model f with scalar output is randomly initialized, and the 1-Lipschitz continuous activation function σ within f satisfies $|\sigma(x)| \leq |x|$. Then, given any two empirical distributions P, P' on dataset S, let $\boldsymbol{\Theta}_0, \boldsymbol{\Theta}_0'$ be their corresponding NTKs, denote $Z \triangleq \int \|P(\boldsymbol{x}) - P'(\boldsymbol{x})\| \, \mathrm{d}\boldsymbol{x}$, as $n \to \infty$,*

$$\frac{1}{m}\left| \|\boldsymbol{\Theta}_0\|_{\mathrm{tr}} - \|\boldsymbol{\Theta}_0'\|_{\mathrm{tr}} \right| \leq n_0^{-1} Z$$

with probability arbitrarily close to 1.

Let $\mathcal{M}_{\mathrm{Trace}}$ and $\mathcal{M}'_{\mathrm{Trace}}$ be evaluated on distribution P and P', respectively. Denote $\mathcal{M}_{\min} \triangleq \min(\mathcal{M}_{\mathrm{Trace}}, \mathcal{M}'_{\mathrm{Trace}})$, based on the definition of $\mathcal{M}_{\mathrm{Trace}}$, we have

$$\begin{aligned} \left|\mathcal{M}_{\mathrm{Trace}} - \mathcal{M}'_{\mathrm{Trace}}\right| &= \left| \frac{\mathcal{M}_{\mathrm{Trace}}^2 - (\mathcal{M}'_{\mathrm{Trace}})^2}{\mathcal{M}_{\mathrm{Trace}} + \mathcal{M}'_{\mathrm{Trace}}} \right| \\ &\leq \frac{m^{-1}\left| \|\boldsymbol{\Theta}_0\|_{\mathrm{tr}} - \|\boldsymbol{\Theta}_0'\|_{\mathrm{tr}} \right|}{2\mathcal{M}_{\min}} \\ &\leq (2n_0 \mathcal{M}_{\min})^{-1} Z \, . \end{aligned} \tag{D.40}$$

Let $\mathcal{M}$ and $\mathcal{M}'$ be any metric from Sec. 7.2.2 that are evaluated on the distribution P and P', respectively. Following our Theorem 7.1 and the result above, there exists a constants α such that the following holds with high probability,

$$\mathcal{M}' \leq \alpha \mathcal{M}'_{\mathrm{Trace}} \leq \alpha \left(\mathcal{M}_{\mathrm{Trace}} + (2n_0 \mathcal{M}_{\min})^{-1} Z) \right) . \tag{D.41}$$

Assume that $\mathcal{M}' > \alpha(2n_0\mathcal{M}_{\min})^{-1}Z)$, we further have

$$\frac{1}{\mathcal{M}_{\text{Trace}}} \leq \frac{\alpha}{\mathcal{M}' - \alpha(2n_0\mathcal{M}_{\min})^{-1}Z)} \tag{D.42}$$

By introducing the result above into (D.33), we have

$$\begin{aligned}
\mathcal{L}_{\mathcal{D}}(f_t) &\leq \mathcal{L}_S(f_t) + \frac{2\beta\kappa\sqrt{m\lambda_0}}{\mathcal{M}_{\text{Trace}}} + \frac{2\beta c}{\sqrt{n}} + 3\sqrt{\frac{\log(2/\delta)}{2m}} \\
&\leq \mathcal{L}_S(f_t) + \frac{2\beta\kappa\sqrt{m\lambda_0}/\alpha}{\mathcal{M}' - \alpha(2n_0\mathcal{M}_{\min})^{-1}Z} + \frac{2\beta c}{\sqrt{n}} + 3\sqrt{\frac{\log(2/\delta)}{2m}} \\
&\leq \mathcal{L}_S(f_t) + \mathcal{O}\left(\frac{\kappa}{\mathcal{M}' - \alpha(2n_0\mathcal{M}_{\min})^{-1}Z}\right),
\end{aligned} \tag{D.43}$$

where the last inequality is based on the definition of $\mathcal{O}(\cdot)$. Our proof of Theorem 7.3 hence is concluded.

D.1.5 Proof of Theorem 7.4

Let $\mathbf{W}_{j\cdot}^{(i)}$ denote the j-th row of matrix $\mathbf{W}^{(i)}$, we can compute the gradient of $\mathbf{W}_{j\cdot}^{(i)}$ (represented as a column vector) from function f and f' as below based on the formulation of these two functions in Sec. 7.3.5.

$$\begin{aligned}
\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f(\boldsymbol{x}) &= \boldsymbol{x} \\
\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f'(\boldsymbol{x}) &= \left(\prod_{k'=1}^{i-1} \mathbf{W}^{(k')}\boldsymbol{x}\right)\mathbf{1}^\top \left(\prod_{k=i+1}^{L} \mathbf{W}^{(k)}\right)_{\cdot j},
\end{aligned} \tag{D.44}$$

where $\left(\prod_{k=i+1}^{L} \mathbf{W}^{(k)}\right)_{\cdot j}$ is defined as the j-th column of matrix $\left(\prod_{k=i+1}^{L} \mathbf{W}^{(k)}\right)$, i.e.,

$$\left(\prod_{k=i+1}^{L} \mathbf{W}^{(k)}\right)_{\cdot j} \triangleq \left(\mathbf{W}^{(i+1)}\cdots\mathbf{W}^{(L)}\right)_{\cdot j} = \mathbf{W}^{(L)}\mathbf{W}^{(L-1)}\cdots\mathbf{W}_{\cdot j}^{(i+1)}, \tag{D.45}$$

Consequently, the NTK matrix of initialized wide architecture can be

represented as

$$\begin{aligned}\boldsymbol{\Theta}_0(\boldsymbol{x},\boldsymbol{x}') &= \sum_{i=1}^{L}\sum_{j=1}^{n}\left(\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f(\boldsymbol{x})\right)^{\top}\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f(\boldsymbol{x}) \\ &= \sum_{i=1}^{L}\sum_{j=1}^{n}\boldsymbol{x}^{\top}\boldsymbol{x}' = nL\cdot\boldsymbol{x}^{\top}\boldsymbol{x}' .\end{aligned} \tag{D.46}$$

Meanwhile, the NTK matrix of initialized deep architecture can be represented as

$$\begin{aligned}\boldsymbol{\Theta}_0'(\boldsymbol{x},\boldsymbol{x}') &= \sum_{i=1}^{L}\sum_{j=1}^{n}\left(\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f'(\boldsymbol{x})\right)^{\top}\nabla_{\mathbf{W}_{j\cdot}^{(i)}} f'(\boldsymbol{x}) \\ &= \sum_{i=1}^{L}\sum_{j=1}^{n}\left(\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\boldsymbol{x}\right)\mathbf{1}^{\top}\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^{\top}\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\boldsymbol{x}'\right)\mathbf{1}^{\top}\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j} \\ &= \sum_{i=1}^{L}\sum_{j=1}^{n}\left(\mathbf{1}^{\top}\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^{2}\boldsymbol{x}^{\top}\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)^{\top}\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)\boldsymbol{x}' \\ &= \boldsymbol{x}^{\top}\sum_{i=1}^{L}\sum_{j=1}^{n}\left(\mathbf{1}^{\top}\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^{2}\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)^{\top}\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)\boldsymbol{x}' .\end{aligned} \tag{D.47}$$

Since each element in $\mathbf{W}^{(i)}$ is initialized using standard normal distribution, we have following simplified expectation by exploring the fact

that $\mathbb{E}\left[\mathbf{W}^{(i)}\right] = \mathbf{0}\mathbf{0}^\top$ and $\mathbb{E}\left[\left(\mathbf{W}^{(i)}\right)^\top \mathbf{W}^{(i)}\right] = n\mathbf{I}$.

$$\begin{aligned}
\mathbb{E}\left[\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)^\top \prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right] &= \mathbb{E}\left[\left(\mathbf{W}^{(1)}\right)^\top \cdots \left(\mathbf{W}^{(i-1)}\right)^\top \mathbf{W}^{(i-1)} \cdots \mathbf{W}^{(1)}\right] \\
&= \mathbb{E}\left[\left(\mathbf{W}^{(1)}\right)^\top \mathbb{E}\left[\cdots \mathbb{E}\left[\left(\mathbf{W}^{(i-1)}\right)^\top \mathbf{W}^{(i-1)}\right]\cdots\right]\mathbf{W}^{(1)}\right] \\
&= \mathbb{E}\left[\left(\mathbf{W}^{(1)}\right)^\top \mathbb{E}\left[\cdots \mathbb{E}\left[\left(\mathbf{W}^{(i-2)}\right)^\top (n\mathbf{I})\,\mathbf{W}^{(i-2)}\right]\cdots\right]\mathbf{W}^{(1)}\right] \\
&= n^{i-1}\mathbf{I}\,.
\end{aligned} \tag{D.48}$$

Similarly, we also have

$$\begin{aligned}
\mathbb{E}\left[\left(\mathbf{1}^\top\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^2\right] &= \mathbf{1}^\top\mathbb{E}\left[\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\left(\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^\top\right]\mathbf{1} \\
&= \mathbf{1}^\top\mathbb{E}\left[\mathbf{W}^{(L)}\mathbb{E}\left[\cdots\mathbb{E}\left[\mathbf{W}_{\cdot j}^{(i+1)}\left(\mathbf{W}_{\cdot j}^{(i+1)}\right)^\top\right]\cdots\right]\left(\mathbf{W}^{(L)}\right)^\top\right]\mathbf{1} \\
&= \mathbf{1}^\top\mathbb{E}\left[\mathbf{W}^{(L)}\mathbb{E}\left[\cdots\mathbb{E}\left[\mathbf{W}^{(i+2)}\mathbf{I}\left(\mathbf{W}^{(i+2)}\right)^\top\right]\cdots\right]\left(\mathbf{W}^{(L)}\right)^\top\right]\mathbf{1} \\
&= \mathbf{1}^\top\mathbb{E}\left[\mathbf{W}^{(L)}\mathbb{E}\left[\cdots\mathbb{E}\left[\mathbf{W}^{(i+3)}n\mathbf{I}\left(\mathbf{W}^{(i+3)}\right)^\top\right]\cdots\right]\left(\mathbf{W}^{(L)}\right)^\top\right]\mathbf{1} \\
&= n^{l-i-1}\mathbf{1}^\top\mathbf{1} \\
&= n^{L-i}\,.
\end{aligned} \tag{D.49}$$

Since $\mathbf{W}^{(i)}$ in each layer is initialized independently, we achieve the following result by introducing the equality above and expectation over

model parameters into (D.44).

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{\Theta}_0'(\boldsymbol{x},\boldsymbol{x}')\right] &= \boldsymbol{x}^\top \mathbb{E}\left[\sum_{i=1}^{L}\sum_{j=1}^{n}\left(\mathbf{1}^\top\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^2\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)^\top\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)\right]\boldsymbol{x}' \\
&= \boldsymbol{x}^\top\left(\sum_{i=1}^{L}\sum_{j=1}^{n}\mathbb{E}\left[\left(\mathbf{1}^\top\left(\prod_{k=i+1}^{L}\mathbf{W}^{(k)}\right)_{\cdot j}\right)^2\right]\mathbb{E}\left[\left(\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right)^\top\prod_{k'=1}^{i-1}\mathbf{W}^{(k')}\right]\right)\boldsymbol{x}' \\
&= \boldsymbol{x}^\top\left(\sum_{i=1}^{L}\sum_{j=1}^{n} n^{L-i}\cdot n^{i-1}\mathbf{I}\right)\boldsymbol{x}' \\
&= Ln^L\boldsymbol{x}^\top\boldsymbol{x}' \,.
\end{aligned} \tag{D.50}$$

Since $\mathbf{X}^\top\mathbf{X} = \mathbf{I}$ with $\mathbf{X} \triangleq [\boldsymbol{x}_1\boldsymbol{x}_2\cdots\boldsymbol{x}_m]$, we finally conclude the proof by

$$\begin{aligned}
\boldsymbol{\Theta}_0(\mathbf{X},\mathbf{X}) &= Ln\cdot\mathbf{I} \\
\mathbb{E}\left[\boldsymbol{\Theta}_0'(\mathbf{X},\mathbf{X})\right] &= Ln^L\cdot\mathbf{I}\,.
\end{aligned} \tag{D.51}$$