UNIVERSITY OF CALIFORNIA

Los Angeles

Advancing Automated Machine Learning:
Neural Architectures and Optimization Algorithms

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Xiangning Chen

2023

ABSTRACT OF THE DISSERTATION

Advancing Automated Machine Learning:
Neural Architectures and Optimization Algorithms

by

Xiangning Chen
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2023
Professor Cho-Jui Hsieh, Chair

The field of Automated Machine Learning (AutoML) has gained immense attention for its ability to automate complex machine learning tasks, yet it is still an evolving discipline requiring nuanced approaches to be fully realized. This thesis, "Advancing Automated Machine Learning: Neural Network Architectures and Optimization Algorithms," provides a comprehensive investigation into two foundational pillars: Neural Architecture Search (NAS) and optimization algorithms.

In the first half of the thesis, we confront the inherent challenges of stability and robustness in NAS, enhancing its reliability through a perturbation-based regularization scheme. This allows for more consistent and dependable architecture choices. Furthermore, we extend the traditional paradigms of NAS by framing it as a distribution learning problem, and additionally, by applying it to collaborative filtering. These extensions not only broaden the applicability of NAS but also lead to marked improvements in the efficiency and accuracy of recommendation systems.

The latter part of the thesis focuses on the role of optimization in achieving high performance, particularly in transformer architectures. We identify a critical optimization gap and propose strategies for its mitigation, emphasizing the necessity of a transition from purely architecture-based search to include optimization techniques. Then we delve into a groundbreaking approach to optimization algorithm design through symbolic program discovery. This framework automatically discover new optimization methods that outperform traditional algorithms, thereby introducing an unprecedented level of automation in the development of optimization techniques. Our developed Lion algorithm has been widely adopted by the community. This not only advances the state-of-the-art in optimization algorithms but also significantly augments the capabilities and reach of AutoML systems.

By addressing these multifaceted challenges in both neural architecture and optimization algorithm design, this thesis presents a coherent, unified contribution to the advancement of Automated Machine Learning. It is hoped that these collective insights serve as a robust foundation for future research in the ever-evolving landscape of AutoML.

The dissertation of Xiangning Chen is approved.

Wei Wang

Kai-Wei Chang

Mani Srivastava

Cho-Jui Hsieh, Committee Chair

University of California, Los Angeles

2023

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Many thanks to my Ph.D. advisors Prof. Cho-Jui Hsieh for his support and providing me with a nurturing and free environment that allowed me to explore my interests and direction freely, including those in the industry. To my parents, your love, belief, and support have been my anchor throughout this journey. Thank you for being my constant source of encouragement and strength. This achievement is as much yours as it is mine. Thank you to my advisors and collaborators: Minhao Cheng, Ruochen Wang, Xiaocheng Tang, Cihang Xie, Mingxing Tan, Boqing Gong, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Yifeng Lu, Quoc V. Le. Your invaluable support and guidance have been instrumental in my journey. I am deeply grateful for your contributions and mentorship.

VITA

2019	B.Eng. (Electronic Engineering) and B.Ec. (Economics), Tsinghua University.
2019–2021	Teaching Assistant, Computer Science Department, UCLA.
2019–present	Research Assistant, Computer Science Department, UCLA.

CHAPTER 1

Introduction

In recent years, Automated Machine Learning (AutoML) has emerged as a transformative discipline that aims to automate the intricate process of constructing machine learning pipelines. The term "pipeline" here refers to a sequence of stages that transforms raw data into actionable insights. Traditionally, this involves multiple steps, including but not limited to, data preprocessing, feature selection, model selection, and optimization. Each of these steps requires considerable expertise and time, making the end-to-end process labor-intensive and error-prone. AutoML, therefore, seeks to democratize machine learning by lowering the barrier to entry and facilitating the development of robust, high-performing models.

Standing at the forefront of these developments is Neural Architecture Search (NAS), an innovation that marks a watershed moment in the domain of AutoML. Gone are the days when each architectural element of a neural network model had to be intricately crafted by the hands of domain experts. NAS brings a sea change by introducing a layer of automation that revolutionizes this practice. Leveraging a portfolio of sophisticated search algorithms—ranging from reinforcement learning and evolutionary algorithms to Differentiable Architecture Search (DARTS)—NAS has the capability to explore the exponentially large design space of neural network architectures autonomously. This revolution in automation confers dual benefits: it relieves machine learning professionals from the arduous task of model design, allowing them to channel their expertise into defining the problem at hand and enhancing data quality; secondly, it has the potential to significantly shorten the innovation cycles, enabling faster adaptations and implementations of machine learning solutions.

It is straightforward to search by reinforcement learning (Zhong et al., 2018; Zoph and Le, 2017; Zoph et al., 2018) and evolutionary algorithm (Liu et al., 2017; Miikkulainen et al., 2019; Real et al., 2017; Stanley and Miikkulainen, 2002) due to the discrete nature of the architecture space. However, these methods usually require massive computation resources. A variety of approaches are then proposed to reduce the search cost including one-shot architecture search (Bender et al., 2018; Brock et al., 2018; Pham et al., 2018), performance estimation (Klein et al., 2017) and network morphisms (Cai et al., 2018a,b; Elsken et al., 2019). For example, one-shot architecture search methods construct a super-network covering all candidate architectures, where sub-networks with shared components also share the corresponding weights. Then the super-network is trained only once, which is much more efficient. As a particularly popular instance of one-shot methods, DARTS (Liu et al., 2018b) enables the search process to be performed with a gradient-based optimizer in an end-to-end manner. It applies continuous relaxation that transforms the categorical choice of architectures into continuous architecture parameters. The resulting supernet can be optimized via gradient-based methods, and the operations associated with the largest architecture parameters are selected to form the final architecture.

Despite being computationally efficient, the stability and generalizability of DARTS have been challenged recently. Many (Yu et al., 2020; Zela et al., 2020b) have observed that although the validation accuracy of the mixture architecture keeps growing, the performance of the derived architecture collapses when evaluation. Such instability makes DARTS converge to distorted architectures. For instance, Chu et al. (2019) and Liang et al. (2019) find that parameter-free operations such as *skip connection* dominate the generated architecture, and DARTS has a preference towards wide and shallow structures (Shu et al., 2020). To alleviate this issue, some (Liang et al., 2019; Zela et al., 2020b) propose to early stop the search process based on handcrafted criteria. However, the inherent instability starts from the very beginning and early stopping is a compromise without actually improving the search algorithm. This precarious balance of efficiency and reliability underscores the need for deeper investigations aimed at stabilizing the search methods, making them more predictable and reliable for broader adoption. The first half of the thesis introduces two novel approaches that enhance the reliability and robustness of differentiable Neural Architecture Search (NAS) methods. These approaches are built upon the DARTS framework and employ techniques such as perturbation-based regularization and architecture distribution learning. Additionally, the first half showcases the practical application of the newly proposed robust NAS techniques in the domains of recommender systems and knowledge graphs.

The latter portion of the thesis pivots its focus towards the automated discovery of optimization algorithms, revealing a paradigm shift in machine learning research. This transition comes in the backdrop of years of academic and industrial investment in developing novel neural network architectures, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These architectures have demonstrated their effectiveness in various machine learning applications, spanning from computer vision to natural language processing. However, with the advent of the Transformer architecture, the academic discourse has noticeably shifted. No longer is the conversation primarily about building entirely new architectures; instead, the focus has shifted towards refining and optimizing existing ones, especially Transformers. This shift is not merely academic; it represents a broader transformation in machine learning priorities. While architectures like CNNs and RNNs still find applications across numerous fields, the Transformer has been instrumental in shaping the current wave of research in large-scale language and multimodal models. This newfound focus has driven a surge in research into ancillary features that bolster these architectures, with optimizers emerging as an area of renewed scrutiny and innovation.

Several handcrafted optimizers have been introduced over the past few years, notably adaptive algorithms that have shown promise in various contexts (Anil et al., 2020; Balles and Hennig, 2018; Bernstein et al., 2018; Dozat, 2016; Liu et al., 2020; Zhuang et al., 2020). However, it's telling that Adam (Kingma and Ba, 2014), especially when augmented with decoupled weight decay to form AdamW (Loshchilov and Hutter, 2019), along with Adafactor featuring factorized second moments (Shazeer and Stern, 2018), continue to be the optimizers of choice for training state-of-the-art models in language (Brown et al., 2020; Devlin et al., 2019; Vaswani et al., 2017), vision (Dai et al., 2021; Dosovitskiy et al., 2021b; Zhai et al., 2021), and in emerging multimodal paradigms (Radford et al., 2021; Saharia et al., 2022; Yu et al., 2022).

Adding another layer of complexity, some researchers are exploring the feasibility of automatically discovering optimization algorithms. The Learning-to-Optimize (L2O) framework Chen et al. (2021a) has been at the forefront of this endeavor, attempting to train parameterized models—typically neural networks—to autonomously generate update rules for optimization (Andrychowicz et al., 2016; Li and Malik, 2017; Metz et al., 2019, 2022). Yet, these models, often designed as black-box optimizers, have demonstrated limited generalizability, particularly when applied to training larger models or longer training steps. Parallel to this, another subset of research employs techniques like reinforcement learning and Monte Carlo Sampling to automate the discovery process (Bello et al., 2017; Wang et al., 2022). However, these methods usually confine their search within predefined boundaries, often limiting the search space to predefined operands (such as gradients and momentum) and operators (like unary and binary mathematical operations). As a result, these constrained approaches usually fall short of unlocking new potential, such as modifying how momentum is tracked or integrated into the update mechanism.

Drawing inspiration from the ambitious AutoML-Zero project (Real et al., 2020), which aims to search every component of a machine learning pipeline, the latter half of this thesis aspires to contribute significantly to the automated discovery of optimization algorithms. Our motivation is particularly centered around the potential for improving the optimization landscape of Transformer architectures, which have shown tendencies to converge to sharp, suboptimal local minima.

To this end, we introduce a novel method that formulates the task of algorithm discovery as a program search. This approach culminates in the development of a new, highly effective optimization algorithm that we have named Lion, an acronym standing for *Evo**L**ved S**i**gn M**o**me**n**tum*. Unlike many prevalent adaptive algorithms that track various parameters, Lion simplifies the process by focusing solely on momentum tracking and leveraging the sign operation to compute updates. This results in a model that requires less memory overhead and produces consistent update magnitudes across all dimensions, thus offering a compelling new direction in the quest for optimization algorithms better suited to modern neural architectures.

CHAPTER 2

Stabilize and Robustify Neural Architecture Search

In recent years, Neural Architecture Search (NAS) has attracted lots of attentions for its potential to democratize deep learning. For a practical end-to-end deep learning platform, NAS plays a crucial role in discovering task-specific architecture depending on users' configurations (e.g., dataset, evaluation metric, etc.). Pioneers in this field develop prototypes based on reinforcement learning (Zoph and Le, 2017), evolutionary algorithms (Real et al., 2019a) and Bayesian optimization (Liu et al., 2018a). These works usually incur large computation overheads, which make them impractical to use. More recent algorithms significantly reduce the search cost including one-shot methods (Bender et al., 2018; Pham et al., 2018), a continuous relaxation of the space (Liu et al., 2018b) and network morphisms (Cai et al., 2018a). In particular, Liu et al. (2018b) proposes a differentiable NAS framework - DARTS, converting the categorical operation selection problem into learning a continuous architecture mixing weight. They formulate a bi-level optimization objective, allowing the architecture search to be efficiently performed by a gradient-based optimizer.

2.1 Problem Settings

Cell-Based Search Space The cell-based search space is constructed by replications of normal and reduction cells (Liu et al., 2018b; Zoph et al., 2018). A normal cell keeps the spatial resolution while a reduction cell halves it but doubles the number of channels. Every cell is represented by a DAG with N nodes and E edges, where every node represents a latent representation $\mathbf{x}^i$ and every edge (i, j) is associated with an operations $o^{(i,j)}$ (e.g., *max pooling* or *convolution*) selected from a predefined candidate space $\mathcal{O}$. The output of a node is a summation of all input flows, i.e., $\mathbf{x}^j = \sum_{i<j} o^{(i,j)}(\mathbf{x}^i)$, and a concatenation of intermediate node outputs, i.e., $concat(\mathbf{x}^2, ..., \mathbf{x}^{N-1})$, composes the cell output, where the first two input nodes $\mathbf{x}^0$ and $\mathbf{x}^1$ are fixed to be the outputs of previous two cells.

Gradient-Based Search via Continuous Relaxation To enable gradient-based optimization, Liu et al. (2018b) apply a continuous relaxation to the discrete space. Concretely, the information passed from node i to node j is computed by a weighted sum of all operations alone the edge, forming a mixed-operation $\hat{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \theta_o^{(i,j)} o(x)$. The operation mixing weight $\theta^{(i,j)}$ is defined over the probability simplex and its magnitude represents the strength of each operation. Therefore, the architecture search can be cast as selecting the operation associated with the highest mixing weight for each edge.

Bilevel-Optimization with Simplex Constraints With continuous relaxation, the network weight w and operation mixing weight θ can be jointly optimized by solving a constraint bi-level optimization problem:

$$\min_{\theta} \ \mathcal{L}_{val}(w^*, \theta) \quad \text{s.t.} \ \ w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \theta), \qquad \sum_{o=1}^{|\mathcal{O}|} \theta_o^{(i,j)} = 1, \ \forall \ (i,j), \ i < j, \tag{2.1}$$

where the simplex constraint $\sum_{o=1}^{|\mathcal{O}|} \theta_o^{(i,j)} = 1$ can be either solved explicitly via Lagrangian function (Li et al., 2020), or eliminated by substitution method (e.g., $\theta = Softmax(\alpha), \alpha \in \mathcal{R}^{|\mathcal{O}| \times |E|}$) (Liu et al., 2018b).

2.2 Performance Collapse of DARTS

While current differentiable Neural Architecture Search (NAS) methods, such as DARTS, have shown promising results, they exhibit several limitations that constrain their practical applicability.

First, the stability and generalizability of DARTS are matters of ongoing debate. Numerous studies (Yu et al., 2020; Zela et al., 2020b) have noted that while the validation accuracy for the mixed architecture consistently improves, the performance of the resulting architecture deteriorates upon evaluation. This inconsistency leads DARTS to converge on flawed architectures. For example, works by Chu et al. (2019) and Liang et al. (2019) reveal that parameter-free operations like *skip connection* overwhelmingly influence the architecture that DARTS generates, creating a bias toward wide and shallow structures (Shu et al., 2020). To mitigate this problem, some researchers (Liang et al., 2019; Zela et al., 2020b) recommend prematurely terminating the search process based on manually defined criteria. However, this approach merely sidesteps the underlying issue without enhancing the robustness of the search algorithm itself.

Second, there is a discrepancy between the search and evaluation phases of DARTS. During the search phase, proxy tasks are often utilized with smaller datasets or reduced-complexity networks, largely due to the high memory requirements of differentiable NAS. This practice further complicates the task of obtaining architectures that are both efficient and effective in real-world settings.

2.3 Stabilizing Neural Architecture Search via Perturbation-based Regularization

The stability and generalizability of the DARTS algorithm have been challenged for yielding deteriorating architectures as the search proceeds. We find that the precipitous validation loss landscape, which leads to a dramatic performance drop when distilling the final architecture, is an essential factor that causes instability. Based on this observation, we propose a perturbation-based regularization, named SmoothDARTS (SDARTS), to smooth the loss landscape and improve the generalizability of DARTS. In particular, our new formulations stabilize DARTS by either random smoothing or adversarial attack. The search trajectory on NAS-Bench-

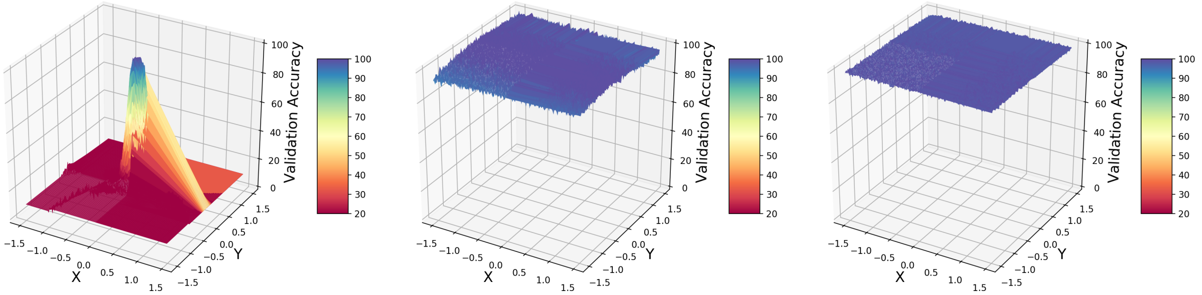


Figure 2.1: The landscape of validation accuracy regarding the architecture weight A on CIFAR-10 for DARTS (Left), SDARTS-RS (Middle), and SDARTS-ADV (Right). The X-axis is the gradient direction $\nabla_A L_{valid}$, while the Y-axis is another random orthogonal direction (best viewed in color).

1Shot1 demonstrates the effectiveness of our approach and due to the improved stability, we achieve performance gain across various search spaces on four datasets. Furthermore, we mathematically show that SDARTS implicitly regularizes the Hessian norm of the validation loss, which accounts for a smoother loss landscape and improved performance.

2.3.1 Origins of Instability in the DARTS

An important source of the performance collapse in DARTS discussed in Section 2.2 is the final projection step to derive the actual discrete architecture from the continuous mixture architecture. There is often a huge performance drop in this projection step, so the validation accuracy of the mixture architecture, which is optimized by DARTS, may not be correlated with the final validation accuracy.

As shown in Figure 2.1 (Left), DARTS often converges to sharp regions, so small perturbations will dramatically decrease the validation accuracy, let alone the final projection step in DARTS. Moreover, the sharp cone in the landscape illustrates that the network weight w is almost only applicable to the current architecture weight A. Bender et al. (2018) also discovers a similar phenomenon that the shared weight w of the one-shot network is sensitive

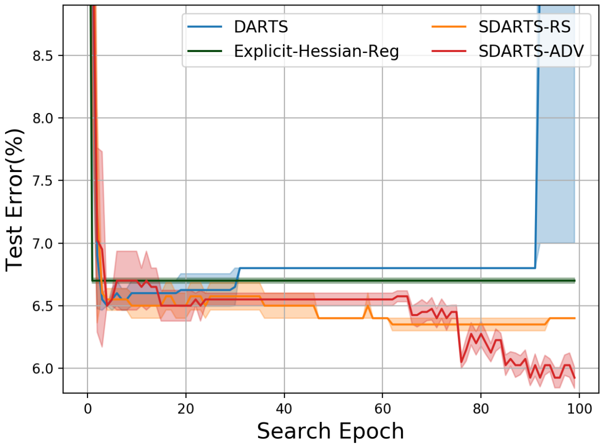


Figure 2.2: Anytime test error (mean ± std) of DARTS, explicit Hessian regularization, SDARTS-RS and SDARTS-ADV on NAS-Bench-1Shot1 (best viewed in color).

and only works for a few sub-networks. This empirically prevents DARTS from fully exploring the architecture space.

To address these problems, we propose two novel formulations. Intuitively, the optimization of A is based on w that performs well on nearby configurations rather than exactly the current one. This leads to smoother landscapes as shown in Figure 2.1 (Middle and Right).

2.3.2 Proposed method

Motivation During the DARTS search procedure, a continuous architecture weight A is used, but it has to be projected to derive the discrete architecture eventually. There is often a huge performance drop in the projection stage, and thus a good mixture architecture does not imply a good final architecture. Therefore, although DARTS can consistently reduce the validation error of the mixture architecture, the validation error after projection is very unstable and could even blow up, as shown in Figure 2.2 and 2.3.

This phenomenon has been discussed in several recent papers (Liang et al., 2019; Zela et al., 2020b), and Zela et al. (2020b) empirically finds that the instability is related to

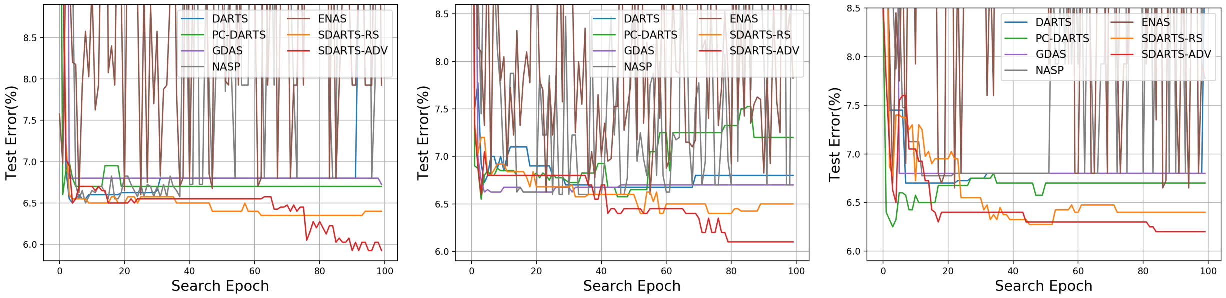


Figure 2.3: Anytime test error on NAS-Bench-1Shot1 (best viewed in color). From left to right: Spaces 1 through 3.

the norm of Hessian $\nabla_A^2 L_{\text{valid}}$. To verify this phenomenon, we plot the validation accuracy landscape of DARTS in Figure 2.1 (Left), which is extremely sharp – small perturbation on A can hugely reduce the validation accuracy from over 90% to less than 10%. This also undermines DARTS' ability to explore the architecture space: A can only change slightly at each iteration because the current w only works within a small local region.

Proposed Formulation To address this issue, intuitively we want to force $L_{\text{val}}(\bar{w}(A), A+\Delta)$ to be more smooth with respect to the perturbation Δ. This leads to the following two versions of SDARTS by redefining $\bar{w}(A)$:

$$\begin{aligned} &\min_A L_{\text{val}}(\bar{w}(A), A), \text{ s.t.} \\ &\text{SDARTS-RS: } \bar{w}(A) = \arg\min_w E_{\delta \sim U_{[-\epsilon,\epsilon]}} L_{\text{train}}(w, A+\delta) \\ &\text{SDARTS-ADV: } \bar{w}(A) = \arg\min_w \max_{\|\delta\| \leq \epsilon} L_{\text{train}}(w, A+\delta) \end{aligned} \tag{2.2}$$

where $U_{[-\epsilon,\epsilon]}$ represents the uniform distribution between $-\epsilon$ and ϵ. The main idea is that instead of using w that only performs well on the current A, we replace it by the $\bar{w}$ defined in (2.2) that performs well within a neighborhood of A. This forces our algorithms to focus on $(\bar{w}, A)$ pairs with smooth loss landscapes. For SDARTS-RS, we set $\bar{w}$ as the minimizer of the expected loss under small random perturbation bounded by ϵ. This is based on the idea of random smoothing, which randomly averaging the neighborhood of a given function to obtain a smoother version (Cohen et al., 2019; Lecuyer et al., 2019). On the other hand, we set $\bar{w}$ to minimize the worst-case training loss under small perturbation of ϵ for SDARTS-ADV. This is based on the idea of adversarial training, which is a widely used technique in adversarial defense (Madry et al., 2018a).

Algorithm 1 Training of SDARTS

```
Generate a mixed operation $\bar{o}^{(i,j)}$ for every edge $(i, j)$
while not converged do
    Update architecture $A$ by descending $\nabla_A L_{val}(w, A)$
    Compute $\delta$ based on equation (2.3) or (2.4)
    Update weight $w$ by descending $\nabla_w L_{train}(w, A + \delta)$
end while
```

2.3.3 Search Algorithms

The optimization algorithm for solving the proposed formulations is described in Algorithm 1. Similar to DARTS, our algorithm is based on alternating minimization between A and w. For SDARTS-RS, $\bar{w}$ is the minimizer of the expected loss altered by a randomly chosen δ, which can be optimized by SGD directly. We sample the following δ and add it to A before running a single step of SGD on w [1]:

$$\delta \sim U_{[-\epsilon,\epsilon]}. \tag{2.3}$$

This approach is very simple (adding only one line of the code) and efficient (doesn't introduce any overhead), and we find that it is quite effective to improve the stability. As shown in Figure 2.1 (Middle), the sharp cone disappears and the landscape becomes much smoother, which maintains high validation accuracy under perturbation on A.

[1] We use uniform random for simplicity, while in practice the approach works also with other random perturbations, such as Gaussian.

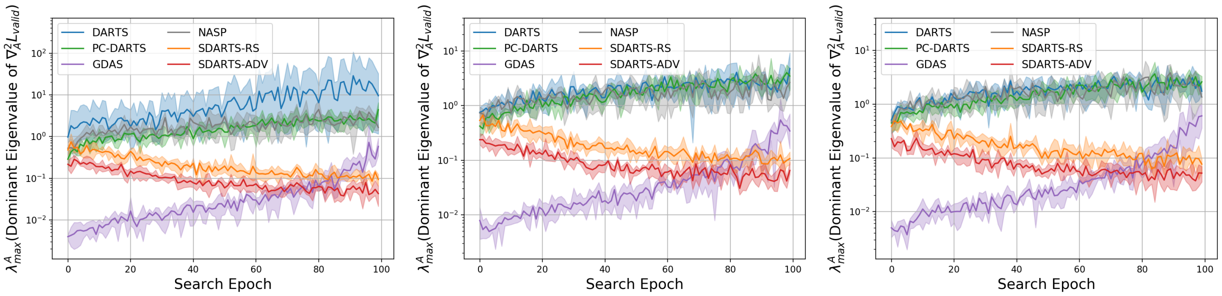


Figure 2.4: Trajectory (mean ± std) of the Hessian norm on NAS-Bench-1Shot1 (best viewed in color). From left to right: Spaces 1 through 3.

For SDARTS-ADV, we consider the worst-case loss under certain perturbation level, which is a stronger requirement than the expected loss in SDARTS-RS. The resulting landscape is even smoother as illustrated in Figure 2.1 (Right). In this case, updating $\bar{w}$ needs to solve a min-max optimization problem beforehand. We employ the widely used multi-step projected gradient descent (PGD) on the negative training loss to iteratively compute δ:

$$\delta^{n+1} = \mathcal{P}(\delta^n + lr * \nabla_{\delta^n} L_{\text{train}}(w, A + \delta^n)) \tag{2.4}$$

where $\mathcal{P}$ denotes the projection onto the chosen norm ball (e.g. clipping in the case of the ℓ_∞ norm) and lr denotes the learning rate.

In the next section, we will mathematically explain why SDARTS-RS and SDARTS-ADV improve the stability and generalizability of DARTS.

2.3.4 Implicit Regularization on Hessian Matrix

It has been empirically pointed out in (Zela et al., 2020b) that the dominant eigenvalue of $\nabla^2_A L_{\text{val}}(w, A)$ (spectral norm of Hessian) is highly correlated with the generalization quality of DARTS solutions. In standard DARTS training, the Hessian norm usually blows up, which leads to deteriorating (test) performance of the solutions. In Figure 2.4, we plot this Hessian norm during the training procedure and find that the proposed methods, including both SDARTS-RS and SDARTS-ADV, consistently reduce the Hessian norms during the training procedure. In the following, we first explain why the spectral norm of Hessian is correlated with the solution quality, and then formally show that our algorithms can implicitly control the Hessian norm.

Why is Hessian norm correlated with solution quality? Assume (w^*, A^*) is the optimal solution of the original DARTS in the continuous space:

$$\min_A L_{\text{val}}(w^*(A), A), \text{ s.t. } w^* = \arg\min_w L_{\text{train}}(w, A), \tag{2.5}$$

while $\bar{A}$ is the discrete solution by projecting A^* to the simplex. Based on Taylor expansion and assume $\nabla_A L_{\text{val}}(w^*, A^*) = 0$ due to optimality condition, we have

$$L_{\text{val}}(w^*, \bar{A}) = L_{\text{val}}(w^*, A^*) + \frac{1}{2}(\bar{A} - A^*)^T \bar{H}(\bar{A} - A^*) \tag{2.6}$$

where $\bar{H} = \int_{A^*}^{\bar{A}} \nabla_A^2 L_{\text{val}}(w^*, A) dA$ is the average Hessian. If we assume that Hessian is stable in a local region, then the quantity of $C = \|\nabla_A^2 L_{\text{val}}(w^*, A^*)\| \|\bar{A} - A^*\|^2$ can approximately bound the performance drop when projecting A^* to $\bar{A}$ with a fixed w^*. After fine tuning, $L_{\text{val}}(\bar{w}, \bar{A})$ where $\bar{w}$ is the optimal weight corresponding to $\bar{A}$ is expected to be even smaller than $L_{\text{val}}(w^*, \bar{A})$, if the training and validation losses are highly correlated. Therefore, the performance of $L_{\text{val}}(\bar{w}, \bar{A})$, which is the quantity we care, will also be bounded by C. Note that the bound could be quite loose since it assumes the network weight remains unchanged when switching from A^* to $\bar{A}$. A more precise bound can be computed by viewing $g(A) = L_{\text{val}}(w^*(A), A)$ as a function only paramterized by A, and then calculate its derivative/Hessian.

Controlling spectral norm of Hessian is non-trivial. With the observation that the solution quality of DARTS is related to $\|\nabla_A^2 L_{\text{val}}(w^*, A^*)\|$, an immediate thought is to explicitly control this quantity during the optimization procedure. To implement this idea, we add an auxiliary term - the finite difference estimation of Hessian matrix $\nabla_A L_{\text{val}}(A + \epsilon) - \nabla_A L_{\text{val}}(A - \epsilon)$ to the loss function when updating A. However, this requires much additional memory to build a computational graph of the gradient, and Figure 2.2 suggests that it takes some effect compared with DARTS but is worse than both SDARTS-RS and SDARTS-ADV. One potential reason is the high dimensionality – there are too many directions of ϵ to choose from and we can only randomly sample a subset of them at each iteration.

Why can SDARTS-RS implicitly control Hessian? In SDARTS-RS, the objective function becomes

$$
\begin{aligned}
&E_{\delta\sim U_{[-\epsilon,\epsilon]}}L(w, A+\delta) && (2.7)\\
\approx &E_{\delta\sim U_{[-\epsilon,\epsilon]}}\left[L(w,A)+\delta\nabla_A L(w,A)+\frac{1}{2}\delta^T\nabla_A^2 L(w,A)\delta\right] && (2.8)\\
= &L(w,A)+\frac{\epsilon^2}{6}\mathrm{Tr}\left\{\nabla_A^2 L(w,A)\right\} && (2.9)
\end{aligned}
$$

where the second term in (2.8) is canceled out since $E[\delta]=0$ and the off-diagonal elements of the third term becomes 0 after taking the expectation on δ. The update of w in SDARTS-RS can thus implicitly controls the trace norm of $\nabla_A^2 L(w,A)$. If the matrix is close to PSD, this is approximately regularizing the (positive) eigenvalues of $\nabla_A^2 L_{\text{val}}(w,A)$. Therefore, we observe that SDARTS-RS empirically reduces the Hessian norm through its training procedure.

Why can SDARTS-ADV implicitly control Hessian? SDARTS-ADV ensures that the validation loss is small under the worst-case perturbation of A. If we assume the Hessian matrix is roughly constant within ϵ-ball, then adversarial training implicitly minimizes

$$
\begin{aligned}
&\min_{A:\|A-A^*\|\leq\epsilon} L(w,A) && (2.10)\\
\approx &L(w,A^*)+\frac{1}{2}\max_{\|\Delta\|\leq\epsilon}\Delta^T H\Delta && (2.11)
\end{aligned}
$$

when the perturbation is in ℓ_2 norm, the second term becomes the $\frac{1}{2}\epsilon^2\|H\|$, and when the perturbation is in ℓ_∞ norm, the second term is bounded by $\epsilon^2\|H\|$. Thus SDARTS-ADV also approximately minimizes the norm of Hessian. In addition, notice that from (2.10) to (2.11) we assume the gradient is 0, which is the property holds only for A^*. In the intermediate steps for a general A, the stability under perturbation will not only be related to Hessian but also gradient, and in SDARTS-ADV we can still implicitly control the landscape to be smooth by minimizing the first-order term in the Taylor expansion of (2.10).

In the following sections, we first track the anytime performance of our methods on NAS-Bench-1Shot1 in Section 2.3.5, which demonstrates their superior stability and generalizability. Then we perform experiments on the widely used CNN cell space on CIFAR-10 (Section 2.3.6) and RNN cell space on PTB (Section 2.3.7). In Section 2.3.8, we present a detailed comparison between our methods with other popular regularization techniques. At last, we examine the generated architectures and illustrate that our methods mitigate DARTS' bias for certain operations and connection patterns in Section 2.3.9.

2.3.5 Architecture Search on NAS-Bench-1Shot1

Settings NAS-Bench-1Shot1 consists of 3 search spaces based on CIFAR-10, which contains 6,240, 29,160 and 363,648 architectures respectively. The macro architecture of models in all spaces is constructed by 3 stacked blocks, with a *max-pooling* operation in between as the DownSampler. Each block contains 3 stacked cells and the micro architecture of each cell is represented as a DAG. Besides the operation on every edge, the search algorithm also needs to determine the topology of edges connecting input, output nodes and the choice blocks. We refer to their paper (Zela et al., 2020c) for details about the search spaces.

We make a comparison between our methods and state-of-the-art NAS algorithms on all 3 search spaces. We run every NAS algorithm for 100 epochs (twice of the default DARTS setting) to allow a thorough and comprehensive analysis on search stability and generalizability. Hyperparameter settings for 5 baselines are set as their default. For both SDARTS-RS and SDARTS-ADV, the perturbation on A is performed after the softmax layer. We initialize the norm ball ϵ as 0.03 and linearly increase it to 0.3 in all our experiments. The random perturbation δ in SDARTS-RS is sampled uniformly between $-\epsilon$ and ϵ. And we use the 7-step PGD attack under ℓ_∞ norm ball to obtain the δ in SDARTS-ADV. Other settings are the same as DARTS.

To search for 100 epochs on a single NVIDIA GTX 1080 Ti GPU, ENAS, DARTS, GDAS, NASP, PC-DARTS requires 10.5h, 8h, 4.5h, 5h, and 6h respectively. Extra time of SDARTS-RS is just for the random sample, so its search time is approximately the same as DARTS, which is 8h. SDARTS-ADV needs extra steps of forward and backward propagation to perform the adversarial attack, so it spends 16h. Notice that this can be largely reduced by setting the PGD attack step as 1 (FGSM (Goodfellow et al., 2015)), which only brings little performance decrease according to our experiments.

Results We plot the anytime test error averaged from 6 independent runs in Figure 2.3. Also, the trajectory (mean ± std) of the spectral norm of $\nabla_A^2 L_{valid}$ is shown in Figure 2.4. Noting that ENAS is not included in Figure 2.4 since it does not have the architecture weight A. We provide our detailed analysis below.

- DARTS generates architectures with deteriorating performance when the search epoch becomes large, which is in accordance with the observations in (Liang et al., 2019; Zela et al., 2020b). The single-path modifications (GDAS, NASP) take effects to some extent, e.g. GDAS prevents to find worse architectures and remains stable. However, GDAS suffers premature convergence to sub-optimal architectures, and NASP is effective for the first few search epochs before its performance starts to fluctuate like ENAS. A potential reason is that the architecture weight A is clipped to the nearest boundary when it can not satisfy some range constraint. This makes NASP confused when choosing among operations if their corresponding weights are similar on certain edges. The partial channel connection introduced by PC-DARTS makes it the best baseline on Space 1 and 3, but PC-DARTS also suffers severely degenerate performance on Space 2.

- SDARTS-RS outperforms all 5 baselines on 3 search spaces. It better explores the architecture space and meanwhile overcomes the instability issue in DARTS. SDARTS-ADV achieves even better performance by forcing w to minimize the worst-case loss

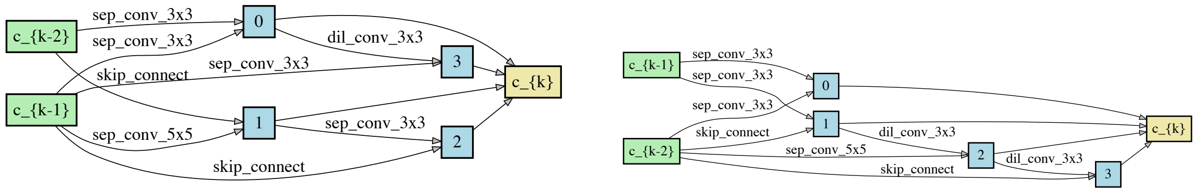


Figure 2.5: Normal cells discovered by SDARTS-RS (Left) and SDARTS-ADV (Right) on CIFAR-10.

around a neighborhood of A. Its anytime test error continues to decrease when the search epoch is larger than 80, which does not occur for any other method.

- As explained in Section 2.3.4, the spectral norm λ_{max}^{A} of Hessian $\nabla_{A}^{2} L_{valid}$ has strong correlation with the stability and solution quality. Large λ_{max}^{A} leads to poor generalizability and stability. In agreement with the theoretical analysis that our methods keep minimizing λ_{max}^{A} (Section 2.3.4), both SDARTS-RS and SDARTS-ADV anneal λ_{max}^{A} to a low level throughout the search procedure. In comparison, λ_{max}^{A} in all baselines continue to increase and they even enlarge beyond 10 times after 100 search epochs. Though GDAS has the lowest λ_{max}^{A} at the beginning, it suffers the largest growth rate. The partial channel connection in PC-DARTS can not regularize the Hessian norm, it has a similar λ_{max}^{A} trajectory to DARTS and NASP, which supports their comparably unstable performance.

2.3.6 Architecture Search on CNN Standard Space

Settings We employ SDARTS-RS and SDARTS-ADV to search CNN cells on CIFAR-10 following the search space (with 7 operations) in DARTS (Liu et al., 2018b). The macro architecture is obtained by stacking convolution cells for 8 times, and every cell contains $N = 7$ nodes (2 input nodes, 4 intermediate nodes, and 1 output nodes). For the search phase, we train the mixture architecture for 50 epochs, with the 50K CIFAR-10 dataset be equally

Table 2.1: Comparison with state-of-the-art image classifiers on CIFAR-10.

Architecture	**Test Error**	**Params**	**Search Cost**	**Search**
	(%)	**(M)**	**(GPU days)**	**Method**
DenseNet-BC (Huang et al., 2017)[⋆]	3.46	25.6	-	manual
NASNet-A (Zoph et al., 2018)	2.65	3.3	2000	RL
AmoebaNet-A (Real et al., 2019a)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-B (Real et al., 2019a)	2.55 ± 0.05	2.8	3150	evolution
PNAS (Liu et al., 2018a)[⋆]	3.41 ± 0.09	3.2	225	SMBO
ENAS (Pham et al., 2018)	2.89	4.6	0.5	RL
NAONet (Luo et al., 2018b)	3.53	3.1	0.4	NAO
DARTS (1st) (Liu et al., 2018b)	3.00 ± 0.14	3.3	0.4	gradient
DARTS (2nd) (Liu et al., 2018b)	2.76 ± 0.09	3.3	1	gradient
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	2.8	1.5	gradient
GDAS (Dong and Yang, 2019)	2.93	3.4	0.3	gradient
BayesNAS (Zhou et al., 2019)	2.81 ± 0.04	3.4	0.2	gradient
ProxylessNAS (Cai et al., 2019)[†]	2.08	-	4.0	gradient
NASP (Yao et al., 2020b)	2.83 ± 0.09	3.3	0.1	gradient
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	3.6	0.1	gradient
R-DARTS(L2) (Zela et al., 2020b)	2.95 ± 0.21	-	1.6	gradient
SDARTS-RS	2.67 ± 0.03	3.4	0.4[‡]	gradient
SDARTS-ADV	2.61 ± 0.02	3.3	1.3[‡]	gradient

[⋆] Obtained without cutout augmentation.

[†] Obtained on a different space with PyramidNet (Han et al., 2017) as the backbone.

[‡] Recorded on a single GTX 1080Ti GPU.

split into training and validation set. Following Liu et al. (2018b), the network weight w is optimized on the training set by an SGD optimizer with momentum as 0.9 and weight decay as 3×10^{-4}, where the learning rate is annealed from 0.025 to 1e-3 following a cosine schedule. Meanwhile, we use an Adam optimizer with learning rate 3e-4 and weight decay 1e-3 to learn the architecture weight A on the validation set. For the evaluation phase, the macro structure consists of 20 cells and the initial number of channels is set as 36. We train the final architecture by 600 epochs using the SGD optimizer with a learning rate cosine scheduled from 0.025 to 0, a momentum of 0.9 and a weight decay of 3e-4. The drop probability of ScheduledDropPath increases linearly from 0 to 0.2, and the auxiliary tower Zoph and Le (2017) is employed with a weight of 0.4. We also utilize CutOut DeVries and Taylor (2017) as the data augmentation technique and report the result (mean ± std) of 4 independent runs with different random seeds.

Results Table 2.1 summarizes the comparison of our methods with state-of-the-art algorithms, and the searched normal cells are visualized in Figure 2.5. We achieve performance gain compared with DARTS and most of its variants. Moreover, the variance of SDARTS-RS is considerably better than baselines and SDARTS-ADV achieves even better stability. PC-DARTS slightly outperforms our methods but has a higher variance. It warm starts w for the first 15 epochs, and the search epoch is comparably smaller, which may alleviate its instability issue discussed in Section 2.3.5. Nevertheless, when searching on various simplified search spaces across 3 datasets, our methods achieve superior stability and test accuracy compared with PC-DARTS as indicated in Section 2.3.8.

2.3.7 Architecture Search on RNN Standard Space

Settings Besides searching for CNN cells, our methods are applicable to various scenarios such as identifying RNN cells. Following DARTS (Liu et al., 2018b), the RNN search space based on PTB contains 5 candidate functions, i.e. *tanh*, *relu*, *sigmoid*, *identity* and *zero*. The macro architecture of the RNN network is comprised of only a single cell consisting of $N = 12$ nodes. The first intermediate node is manually fixed and the rest nodes are determined by the search algorithm. When searching, we train the RNN network for 50 epochs with sequence length as 35. During evaluation, the final architecture is trained by an SGD optimizer, where the batch size is set as 64 and the learning rate is fixed as 20. These settings are the same as DARTS.

Table 2.2: Comparison with language models on PTB (lower perplexity is better).

Architecture	**Perplexity(%)**		**Params**
	valid	test	**(M)**
LSTM + SE (Yang et al., 2018)*	58.1	56.0	22
NAS (Zoph and Le, 2017)	-	64.0	25
ENAS (Pham et al., 2018)	60.8	58.6	24
DARTS (1st) (Liu et al., 2018b)	60.2	57.6	23
DARTS (2nd) (Liu et al., 2018b)†	58.1	55.7	23
GDAS (Dong and Yang, 2019)	59.8	57.5	23
NASP (Yao et al., 2020b)	59.9	57.3	23
SDARTS-RS	58.7	56.4	23
SDARTS-ADV	58.3	56.1	23

* LSTM + SE has 15 softmax experts.

† We achieve 58.5 for validation and 56.2 for test when training the architecture found by DARTS (2nd) ourselves.

Results The results are shown in Table 2.2. SDARTS-RS achieves a validation perplexity of 58.7 and a test perplexity of 56.4. Meanwhile, SDARTS-ADV achieves a validation perplexity of 58.3 and a test perplexity of 56.1. We outperform other NAS methods with similar model size, which demonstrates the effectiveness of our methods for the RNN space. LSTM + SE obtains better results than us, but it benefits from a handcrafted ensemble structure.

2.3.8 Comparison with Other Regularization

Our methods can be viewed as a way to regularize DARTS (implicitly regularize the Hessian norm of validation loss). In this section, we compare SDARTS-RS and SDARTS-ADV with other popular regularization techniques. The compared baselines are 1) partial channel connection (PC-DARTS (Xu et al., 2020)); 2) ScheduledDropPath (Zoph et al., 2018) (R-

Table 2.3: Comparison with popular regularization techniques (test error (%)). The best method is boldface and underlined while the second best is boldface.

Dataset	Space	liu2018darts	PC-DARTS	DARTS-ES	R-DARTS(DP)	R-DARTS(L2)	SDARTS-RS	SDARTS-ADV
C10	S1	3.84	3.11	3.01	3.11	**2.78**	**2.78**	**2.73**
	S2	4.85	3.02	3.26	3.48	3.31	**2.75**	**2.65**
	S3	3.34	**2.51**	2.74	2.93	**2.51**	2.53	**2.49**
	S4	7.20	3.02	3.71	3.58	3.56	**2.93**	**2.87**
C100	S1	29.46	18.87	28.37	25.93	24.25	**17.02**	**16.88**
	S2	26.05	18.23	23.25	22.30	22.44	**17.56**	**17.24**
	S3	28.90	18.05	23.73	22.36	23.99	**17.73**	**17.12**
	S4	22.85	**17.16**	21.26	22.18	21.94	17.17	**15.46**
SVHN	S1	4.58	2.28	2.72	2.55	4.79	**2.26**	**2.16**
	S2	3.53	2.39	2.60	2.52	2.51	**2.37**	**2.07**
	S3	3.41	2.27	2.50	2.49	2.48	**2.21**	**2.05**
	S4	3.05	2.37	2.51	2.61	2.50	**2.35**	**1.98**

DARTS(DP)); 3) L2 regularization on w (R-DARTS(L2)); 3) early stopping (DARTS-ES (Zela et al., 2020b)).

Settings We perform a thorough comparison on four simplified search spaces proposed in (Zela et al., 2020b) across 3 datasets (CIFAR-10, CIFAR-100, and SVHN). All search spaces utilize the same macro architecture as in Section 2.3.6, the difference is that they only contain a portion of candidate operations: the first space S1 contains 2 popular operators per edge, S2 restricts the set of candidate operations on every edge as $\{3\times3$ *separable convolution*, *skip connection*$\}$, the operation set in S3 is $\{3 \times 3$ *separable convolution*, *skip connection*, *zero*$\}$, and S4 simplifies the set as $\{3 \times 3$ *separable convolution*, *noise*$\}$.

Results in Table 2.3 are obtained by running every method 4 independent times and pick the final architecture based on the validation accuracy (retrain from scratch for a few epochs). Other settings are the same as Section 2.3.6.

Results Our methods achieve substantial performance gains compared with baselines. SDARTS-ADV is the best method for all 12 benchmarks and SDARTS-RS strikes the second

Table 2.4: Proportion of parameter-free operations in normal cells found on CIFAR-10.

Space	liu2018darts	PC-DARTS	DARTS-ES	SDARTS-RS	SDARTS-ADV
S1	1.0	0.5	0.375	0.125	0.125
S2	0.875	0.75	0.25	0.375	0.125
S3	1.0	0.125	1.0	0.125	0.125
S4	0.625	0.125	0.0	0.0	0.0

place on 10 benchmarks. The cell discovered on S3 for CIFAR-10 even achieves higher test accuracy than all the methods in Table 2.1 (except for ProxylessNAS that searches based on PyramidNet).

2.3.9 Examine the Searched Architectures

As pointed out in (Liang et al., 2019; Shu et al., 2020; Zela et al., 2020b), DARTS tends to fall into distorted architectures that converge faster, which is another manifestation of its instability. So here we examine the generated architectures and see whether our methods can overcome such bias.

Proportion of Parameter-Free Operations Many have found out that parameter-free operations such as *skip connection* dominate the generated architecture (Liang et al., 2019; Zela et al., 2020b). Though makes architectures converge faster, excessive parameter-free operations can largely reduce the model's representation capability and bring out low test accuracy. As illustrated in Table 2.4, we also find similar phenomenon when searching by DARTS on 4 simplified search spaces in Section 2.3.8. The proportion of parameter-free operations even becomes 100% on S1 and S3, and DARTS can not distinguish the harmful *noise* operation on S4. PC-DARTS achieves some improvements but is not enough since *noise* still appears. DARTS-ES reveals its effectiveness on S2 and S4 but fails on S3 since all operations found are *skip connection*. We do not show R-DARTS(DP) and R-DARTS(L2) here because their discovered cells are not released. In comparison, both SDARTS-RS and

SDARTS-ADV succeed in controlling the portion of parameter-free operations on all search spaces.

Connection Pattern Shu et al. (2020) demonstrates, from both empirical and theoretical aspects, that DARTS tends to favor wide and shallow cells since they often have smoother loss landscape and faster convergence speed. However, these cells may not generalize better than their narrower and deeper variants (Shu et al., 2020). Follow their definitions (suppose every intermediate node has width c), the best cell generated by our methods on CNN standard space (Section 2.3.6) has width $3c$ and depth 4. In contrast, ENAS has width $5c$ and depth 2, DARTS has width $3.5c$ and depth 3, PC-DARTS has width $4c$ and depth 2. Consequently, we succeed in mitigating the bias of connection pattern.

CHAPTER 3

Neural Architecture Search as Distribution Learning

In the previous section, we proposed a perturbation-based regularization method within the DARTS framework to mitigate sharpness issues. Expanding on this concept, this section generalizes the approach to encompass direct learning of a distribution. By sampling from this distribution, we are able to derive the final candidate architectures, an innovative process we have named Dirichlet Neural Architecture Search (DrNAS).

Inspired by the fact that directly optimizing the architecture mixing weight is equivalent to performing point estimation (MLE/MAP) from a probabilistic perspective, we formulate the differentiable NAS as a distribution learning problem instead, which naturally induces stochasticity and encourages exploration. Making use of the probability simplex property of the Dirichlet samples, DrNAS models the architecture mixing weight as random variables sampled from a parameterized Dirichlet distribution. Optimizing the Dirichlet objective can thus be done efficiently in an end-to-end fashion, by employing the pathwise derivative estimators to compute the gradient of the distribution (Martin Jankowiak, 2018). A straightforward optimization, however, turns out to be problematic due to the uncontrolled variance of the Dirichlet, i.e., too much variance leads to training instability and too little variance suffers from insufficient exploration. In light of that, we apply an additional distance regularizer directly on the Dirichlet concentration parameter to strike a balance between the exploration and the exploitation. We further derive a theoretical bound showing that the constrained distributional objective promotes stability and generalization of architecture search by implicitly controlling the Hessian of the validation error.

Furthermore, to enable a direct search on large-scale tasks, we propose a progressive learning scheme, eliminating the gap between the search and evaluation phases. Based on partial channel connection (Xu et al., 2020), we maintain a task-specific super-network of the same depth and number of channels as the evaluation phase throughout searching. To prevent loss of information and instability induced by partial connection, we divide the search phase into multiple stages and progressively increase the channel fraction via network transformation (Chen et al., 2016). Meanwhile, we prune the operation space according to the learnt distribution to maintain the memory efficiency.

We conduct extensive experiments on different datasets and search spaces to demonstrate DrNAS's effectiveness. Based on the DARTS search space (Liu et al., 2018b), we achieve an average error rate of 2.46% on CIFAR-10, which ranks top amongst NAS methods. Furthermore, DrNAS achieves superior performance on large-scale tasks such as ImageNet. It obtains a top-1/5 error of 23.7%/7.1%, surpassing the previous state-of-the-art (24.0%/7.3%) under the mobile setting. On NAS-Bench-201 (Dong and Yang, 2020), we also set new state-of-the-art results on all three datasets with low variance.

3.1 The Proposed Approach - DrNAS

In this section, we first briefly review differentiable NAS setups and generalize the formulation to motivate distribution learning. We then layout our proposed DrNAS and describe its optimization in section 3.1.1. In section 3.1.2, we provide a generalization result by showing that our method implicitly regularizes the Hessian norm over the architecture parameter. The progressive architecture learning method that enables direct search is then described in section 3.1.5.

3.1.1 Differentiable Architecture Search as Distribution Learning

Learning a Distribution over Operation Mixing Weight Previous differentiable architecture search methods view the operation mixing weight θ as learnable parameters that can be directly optimized (Li et al., 2020; Liu et al., 2018b; Xu et al., 2020). This has been shown to cause θ to overfit the validation set and thus induce large generalization error (Chen and Hsieh, 2020; Zela et al., 2020b,c). We recognize that this treatment is equivalent to performing point estimation (e.g., MLE/MAP) of θ in probabilistic view, which is inherently prone to overfitting (Bishop, 2016; Gelman et al., 2004). Furthermore, directly optimizing θ lacks sufficient exploration in the search space, and thus cause the search algorithm to commit to suboptimal paths in the DAG that converges faster at the beginning but plateaus quickly (Shu et al., 2020).

Based on these insights, we formulate the differentiable architecture search as a distribution learning problem. The operation mixing weight θ is treated as random variables sampled from a learnable distribution. Formally, let $q(\theta|\beta)$ denote the distribution of θ parameterized by β. The bi-level objective is then given by:

$$\min_{\beta} E_{q(\theta|\beta)}\big[\mathcal{L}_{val}(w^*, \theta)\big] + \lambda d(\beta, \hat{\beta}) \quad \text{s.t.} \quad w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \theta). \tag{3.1}$$

where $d(\cdot, \cdot)$ is a distance function. Since θ lies on the probability simplex, we select Dirichlet distribution to model its behavior, i.e., $q(\theta|\beta) \sim Dir(\beta)$, where β represents the Dirichlet concentration parameter. Dirichlet distribution is a widely used distribution over the probability simplex (David M. Blei, 2003; Joo et al., 2019; Kessler et al., 2019; Lee et al., 2020), and it enjoys nice properties that enables gradient-based training (Martin Jankowiak, 2018).

The concentration parameter β controls the sampling behavior of Dirichlet distribution and is crucial in balancing exploration and exploitation during the search phase. Let β_o denote the concentration parameter assign to operation o. When $\beta_o \ll 1$ for most $o = 1 \sim |\mathcal{O}|$, Dirichlet tends to produce sparse samples with high variance, reducing the training stability;

when $\beta_o \gg 1$ for most $o = 1 \sim |\mathcal{O}|$, the samples will be dense with low variance, leading to insufficient exploration. Therefore, we add a penalty term in the objective (3.1) to regularize the distance between β and the anchor $\hat{\beta} = 1$, which corresponds to the symmetric Dirichlet.

In section 3.1.2, we also derive a theoretical bound showing that our formulation additionally promotes stability and generalization of the architecture search by implicitly regularizing the Hessian of validation loss w.r.t. architecture parameters.

Learning Dirichlet Parameters via Pathwise Derivative Estimator Optimizing objective (3.1) with gradient-based methods requires back-propagation through stochastic nodes of Dirichlet samples. The commonly used reparameterization trick does not apply to Dirichlet distribution, therefore we approximate the gradient of Dirichlet samples via pathwise derivative estimators (Martin Jankowiak, 2018)

$$\frac{d\theta_i}{d\beta_j} = -\frac{\frac{\partial F_{Beta}}{\partial \beta_j}(\theta_j|\beta_j, \beta_{tot} - \beta_j)}{f_{Beta}(\theta_j|\beta_j, \beta_{tot} - \beta_j)} \times \left(\frac{\delta_{ij} - \theta_i}{1 - \theta_j}\right) \quad i, j = 1, ..., |\mathcal{O}|, \tag{3.2}$$

where F_{Beta} and f_{Beta} denote the CDF and PDF of beta distribution respectively, δ_{ij} is the indicator function, and β_{tot} is the sum of concentrations. F_{Beta} is the iregularised incomplete beta function, for which its gradient can be computed by simple numerical approximation. We refer to (Martin Jankowiak, 2018) for the complete derivations.

Joint Optimization of Model Weight and Architecture Parameter With pathwise derivative estimator, the model weight w and concentration β can be jointly optimized with gradient descent. Concretely, we draw a sample $\theta \sim Dir(\beta)$ for every forward pass, and the gradients can be obtained easily through backpropagation. Following DARTS (Liu et al., 2018b), we approximate w^* in the lower level objective of (3.1) with one step of gradient descent, and run alternative updates between w^* and β.

Selecting the Best Architecture At the end of the search phase, a learnt distribution of operation mixing weight is obtained. We then select the best operation for each edge by the most likely operation in expectation:

$$o^{(i,j)} = \arg\max_{o \in \mathcal{O}} E_{q(\theta_o^{(i,j)}|\beta^{(i,j)})} \big[\theta_o^{(i,j)}\big]. \tag{3.3}$$

In the Dirichlet case, the expectation term is simply the Dirichlet mean $\frac{\beta_o^{(i,j)}}{\sum_{o'} \beta_{o'}^{(i,j)}}$. Note that under the distribution learning framework, we are able to sample a wide range of architectures from the learnt distribution. This property alone has many potentials. For example, in practical settings where both accuracy and latency are concerned, the learnt distribution can be used to find architectures under resource restrictions in a post search phase.

3.1.2 The implicit Regularization on Hessian

It has been observed that the generalization error of differentiable NAS is highly related to the dominant eigenvalue of the Hessian of validation loss w.r.t. architecture parameter. Several recent works report that the large dominant eigenvalue of $\nabla^2_\theta \tilde{\mathcal{L}}_{val}(w, \theta)$ in DARTS results in poor generalization performance (Chen and Hsieh, 2020; Zela et al., 2020b). Our objective (3.1) is the Lagrangian function of the following constraint objective:

$$\min_\beta E_{q(\theta|\beta)} \big[\mathcal{L}_{val}(w^*, \theta)\big] \quad \text{s.t.} \quad w^* = \arg\min_w \ \mathcal{L}_{train}(w, \theta) \ , \ d(\beta, \hat{\beta}) \le \delta, \tag{3.4}$$

Here we derive an approximated lower bound based on (3.4), which demonstrates that our method implicitly controls this Hessian matrix.

Proposition 1 *Let $d(\beta, \hat{\beta}) = \|\beta - \hat{\beta}\|_2 \le \delta$ and $\hat{\beta} = 1$ in the bi-level formulation (3.4). Let μ denote the mean under the Laplacian approximation of Dirichlet. If $\nabla^2_\mu \tilde{\mathcal{L}}_{val}(w^*, \mu)$ is Positive Semi-definite, the upper-level objective can be approximated bounded by:*

$$E_{q(\theta|\beta)}(\mathcal{L}_{val}(w, \theta)) \gtrsim \tilde{\mathcal{L}}_{val}(w^*, \mu) + \frac{1}{2}(\frac{1}{1+\delta}(1 - \frac{2}{|\mathcal{O}|}) + \frac{1}{|\mathcal{O}|}\frac{1}{1+\delta}) tr\big(\nabla^2_\mu \tilde{\mathcal{L}}_{val}(w^*, \mu)\big) \tag{3.5}$$

with:

$$\tilde{\mathcal{L}}_{val}(w^*, \mu) = \mathcal{L}_{val}(w^*, Softmax(\mu)), \quad \mu_o = \log \beta_o - \frac{1}{|\mathcal{O}|} \sum_{o'} \log \beta_{o'}, \quad o = 1, \ldots, |\mathcal{O}|.$$

which is driven by the Laplacian approximation to the Dirichlet distribution (Akash Srivastava, 2017; MacKay, 1998). The lower bound (3.5) indicates that minimizing the expected validation loss controls the trace norm of the Hessian matrix. Empirically, we observe that DrNAS always maintains the dominant eigenvalue of Hessian at a low level (Section 3.3.6). The detailed proof are shown below in Section 3.1.3.

3.1.3 Proof of Proposition 1

Preliminaries: Before the development of Pathwise Derivative Estimator, Laplace Approximate with Softmax basis has been extensively used to approximate the Dirichlet Distribution (Akash Srivastava, 2017; MacKay, 1998). The approximated Dirichlet distribution is:

$$p(\theta(\mathbf{h})|\beta) = \frac{\Gamma(\sum_o \beta_o)}{\prod_o \Gamma(\beta_o)} \prod_o \theta_o^{\beta_o} g(\mathbf{1}^T \mathbf{h}) \tag{3.6}$$

Where $\theta(\mathbf{h})$ is the softmax-transformed $\mathbf{h}$, $\mathbf{h}$ follows multivariate normal distribution, and $g(\cdot)$ is an arbitrary density to ensure integrability (Akash Srivastava, 2017). The mean μ and diagonal covariance matrix Σ of $\mathbf{h}$ depends on the Dirichlet concentration parameter β:

$$\mu_o = \log \beta_o - \frac{1}{|\mathcal{O}|} \sum_{o'} \log \beta_{o'} \qquad \Sigma_o = \frac{1}{\beta_o}(1 - \frac{2}{|\mathcal{O}|}) + \frac{1}{|\mathcal{O}|^2} \sum_{o'} \frac{1}{\beta_{o'}} \tag{3.7}$$

It can be directly obtained from (3.7) that the Dirichlet mean $\frac{\beta_o}{\sum_{o'} \beta_{o'}} = Softmax(\mu)$. Sampling from the approximated distribution can be down by first sampling from $\mathbf{h}$ and then applying Softmax function to obtain θ. We will leverage the fact that this approximation supports explicit reparameterization to derive our proof.

Proof: Apply the above Laplace Approximation to Dirichlet distribution, the unconstrained upper-level objective in (3.4) can then be written as:

$$
\begin{aligned}
&E_{\theta\sim Dir(\beta)}\big[\mathcal{L}_{val}(w^*,\theta)\big] && (3.8)\\
\approx&E_{\epsilon\sim\mathcal{N}(0,\Sigma)}\big[\mathcal{L}_{val}(w^*,Softmax(\mu+\epsilon))\big] && (3.9)\\
\equiv&E_{\epsilon\sim\mathcal{N}(0,\Sigma)}\big[\tilde{\mathcal{L}}_{val}(w^*,\mu+\epsilon)\big] && (3.10)\\
\approx&E_{\epsilon\sim\mathcal{N}(0,\Sigma)}\big[\tilde{\mathcal{L}}_{val}(w^*,\mu)+\epsilon^T\nabla_\mu\tilde{\mathcal{L}}_{val}(w^*,\mu)+\frac{1}{2}\epsilon^T\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)\epsilon\big] && (3.11)\\
=&\tilde{\mathcal{L}}_{val}(w^*,\mu)+\frac{1}{2}tr\big(E_{\epsilon\sim\mathcal{N}(0,\Sigma)}\big[\epsilon\epsilon^T\big]\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)\big) && (3.12)\\
=&\tilde{\mathcal{L}}_{val}(w^*,\mu)+\frac{1}{2}tr\big(\Sigma\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)\big) && (3.13)
\end{aligned}
$$

In our full objective, we constrain the Euclidean distance between learnt Dirichlet concentration and fixed prior concentration $||\beta-\mathbf{1}||_2\leq\delta$. The covariance matrix Σ of approximated softmax Gaussian can be bounded as:

$$
\begin{aligned}
\Sigma_o&=\frac{1}{\beta_o}(1-\frac{2}{|\mathcal{O}|})+\frac{1}{|\mathcal{O}|^2}\sum_{o'}\frac{1}{\beta_{o'}} && (3.14)\\
&\geq\frac{1}{1+\delta}(1-\frac{2}{|\mathcal{O}|})+\frac{1}{|\mathcal{O}|}\frac{1}{1+\delta} && (3.15)
\end{aligned}
$$

Then (3.8) becomes:

$$
\begin{aligned}
&E_{\theta\sim Dir(\beta)}\big[\mathcal{L}_{val}(w^*,\theta)\big] && (3.16)\\
\approx&\tilde{\mathcal{L}}_{val}(w^*,\mu)+\frac{1}{2}tr\big(\Sigma\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)\big) && (3.17)\\
\geq&\tilde{\mathcal{L}}_{val}(w^*,\mu)+\frac{1}{2}(\frac{1}{1+\delta}(1-\frac{2}{|\mathcal{O}|})+\frac{1}{|\mathcal{O}|}\frac{1}{1+\delta})tr\big(\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)\big) && (3.18)
\end{aligned}
$$

The last line holds when $\nabla_\mu^2\tilde{\mathcal{L}}_{val}(w^*,\mu)$ is positive semi-definite. In Section 3.3.6, we provide an empirical justification for this implicit regularization effect of DrNAS.

3.1.4 Connection to Variational Inference

In this section, we draw a connection between DrNAS and Variational Inference (David M. Blei, 2016). We use w, θ, and β to denote the model weight, operation mixing weight, and

Dirichlet concentration parameters respectively, following the main text. The true posterior distribution can be written as $p(\theta|w, D)$, where $D = \{x_n, y_n\}_{n=1}^N$ is the dataset. Let $q(\theta|\beta)$ denote the variational approximation of the true posterior; and assume that $q(\theta|\beta)$ follows Dirichlet distribution. We follow Joo et al. (2019) to assume a symmetric Dirichlet distribution for the prior $p(\theta)$ as well, i.e., $p(\theta) = Dir(\mathbf{1})$. The goal is to minimize the KL divergence between the true posterior and the approximated form, i.e., $\min_\beta KL(q(\theta|\beta)||p(\theta|w, D))$. It can be shown that this objective is equivalent to maximizing the evidence lower bound as below (David M. Blei, 2016):

$$\mathcal{L}(\beta) = E_{q(\theta|\beta)}\big[\log p(D|\theta, w)\big] - KL(q(\theta|\beta)||p(\theta|w)) \tag{3.19}$$

The upper level objective of the bilevel optimization under variational inference framework is then given as:

$$\min_{\beta} \;\; E_{q(\theta|\beta)}\big[-\log p(D_{valid}|\theta, w^*)\big] + KL(q(\theta|\beta)||p(\theta)) \tag{3.20}$$

Note that eq. (3.20) resembles eq. (3.1) if we use the negative log likelihood as the loss function and replace $d(\cdot, \cdot)$ with KL divergence. In practice, we find that using a simple l2 distance regularization works well across datasets and search spaces.

3.1.5 Progressive Architecture Learning

The GPU memory consumption of differentiable NAS methods grows linearly with the size of operation candidate space. Therefore, they usually use a easier proxy task such as training with a smaller dataset, or searching with fewer layers and number of channels (Cai et al., 2019). For instance, the architecture search is performed on 8 cells and 16 initial channels in DARTS (Liu et al., 2018b). But during evaluation, the network has 20 cells and 36 initial channels. Such gap makes it hard to derive an optimal architecture for the target task (Cai et al., 2019).

PC-DARTS (Xu et al., 2020) proposes a partial channel connection to reduce the memory overheads of differentiable NAS, where they only send a random subset of channels to the mixed-operation while directly bypassing the rest channels in a shortcut. However, their method causes loss of information and makes the selection of operation unstable since the sampled subsets may vary widely across iterations. This drawback is amplified when combining with the proposed method since we learn the architecture distribution from Dirichlet samples, which already injects certain stochasticity. As shown in Table 3.1, when directly applying partial channel connection with distribution learning, the test accuracy of the searched architecture decreases over 3% and 18% on CIFAR-10 and CIFAR-100 respectively if we send only 1/8 channels to the mixed-operation.

To alleviate such information loss and instability problem while being memory-efficient, we propose a progressive learning scheme which gradually increases the fraction of channels that are forwarded to the mixed-operation and meanwhile prunes the operation space based on the learnt distribution. We split the search process into consecutive stages and construct a task-specific super-network with the same depth and number of channels as the evaluation phase at the initial stage. Then after each stage, we increase the partial channel fraction, which means that the super-network in the next stage will be wider, i.e., have more convolution channels, and in turn preserve more information. This is achieved by enlarging every convolution weight with a random mapping function similar to Net2Net (Chen et al., 2016). The mapping function $g : \{1, 2, \ldots, q\} \rightarrow \{1, 2, \ldots, n\}$ with $q > n$ is defined as

$$g(j) = \begin{cases} j & j \leq n \\ \text{random sample from } \{1, 2, \ldots, n\} & j > n \end{cases} \tag{3.21}$$

To widen layer l, we replace its convolution weight $\mathbf{W}^{(l)} \in \mathbb{R}^{Out \times In \times H \times W}$ with a new weight $\mathbf{U}^{(l)}$.

$$\mathbf{U}^{(l)}_{o,i,h,w} = \mathbf{W}^{(l)}_{g(o),g(i),h,w}, \tag{3.22}$$

where Out, In, H, W denote the number of output and input channels, filter height and width respectively. Intuitively, we copy $\mathbf{W}^{(l)}$ directly into $\mathbf{U}^{(l)}$ and fulfill the rest part by choosing randomly as defined in g. Unlike Net2Net, we do not divide $\mathbf{U}^{(l)}$ by a replication factor here

Table 3.1: Test accuracy of the derived architectures when searching on NAS-Bench-201 with different partial channel fraction, where $1/K$ channels are sent to the mixed-operation.

	CIFAR-10		CIFAR-100	
K	**Test Accuracy (%)**	**GPU Memory (MB)**	**Test Accuracy (%)**	**GPU Memory (MB)**
1	94.36 ± 0.00	2437	73.51 ± 0.00	2439
2	93.49 ± 0.28	1583	68.48 ± 0.41	1583
4	92.85 ± 0.35	1159	66.68 ± 3.22	1161
8	91.06 ± 0.00	949	55.11 ± 13.78	949
Ours	94.36 ± 0.00	949	73.51 ± 0.00	949

because the information flow on each edge has the same scale no matter the partial fraction is. After widening the super-network, we reduce the operation space by pruning out less important operations according to the Dirichlet concentration parameter β learnt from the previous stage, maintaining a consistent memory consumption. As illustrated in Table 3.1, the proposed progressive architecture learning scheme effectively discovers high accuracy architectures and retains a low GPU memory overhead.

3.2 Discussions and Relationship to Prior Work

Early methods in NAS usually include a full training and evaluation procedure every iteration as the inner loop to guide the consecutive search (Real et al., 2019a; Zoph and Le, 2017; Zoph et al., 2018). Consequently, their computational overheads are beyond acceptance for practical usage, especially on large-scale tasks.

Differentiable NAS Recently, many works are proposed to improve the efficiency of NAS (Bender et al., 2018; Cai et al., 2018a; Liu et al., 2018b; Mei et al., 2020; Pham et al., 2018; Yao et al., 2020a,b). Amongst them, DARTS (Liu et al., 2018b) proposes a differentiable

NAS framework, which introduces a continuous architecture parameter that relaxes the discrete search space. Despite being efficient, DARTS only optimizes a single point on the simplex every search epoch, which has no guarantee to generalize well after the discretization during evaluation. So its stability and generalization have been widely challenged (Chen and Hsieh, 2020; Li and Talwalkar, 2019; Wang et al., 2021; Zela et al., 2020b). Following DARTS, SNAS (Xie et al., 2019) and GDAS (Dong and Yang, 2019) leverage the gumbel-softmax trick to learn the exact architecture parameter. However, their reparameterization is motivated from reinforcement learning perspective, which is an approximation with softmax rather than an architecture distribution. Besides, their methods require tuning of temperature schedule (Caglar Gulcehre, 2017; Yan et al., 2017). GDAS linearly decreases the temperature from 10 to 1 while SNAS anneals it from 1 to 0.03. In comparison, the proposed method can automatically learn the architecture distribution without the requirement of handcrafted scheduling. BayesNAS (Zhou et al., 2019) applies Bayesian Learning in NAS. Specifically, they cast NAS as model compression problem and use Bayes Neural Network as the super-network, which is difficult to optimize and requires oversimplified approximation. While our method considers the stochasticity in architecture mixing weight, as it is directly related to the generalization of differentiable NAS algorithms (Chen and Hsieh, 2020; Zela et al., 2020b).

Memory overhead When dealing with the large memory consumption of differentiable NAS, previous works mainly restrain the number of paths sampled during the search phase. For instance, ProxylessNAS (Cai et al., 2019) employs binary gates and samples two paths every search epoch. PARSEC (Casale et al., 2019) samples discrete architectures according to a categorical distribution to save memory. Similarly, GDAS (Dong and Yang, 2019) and DSNAS (Hu et al., 2020) both enforce a discrete constraint after the gumbel-softmax reparametrization. However, such discretization manifests premature convergence and cause search instability (Zela et al., 2020c; Zhang et al., 2020). Our experiments in Section 3.3.3 also empirically demonstrate this phenomenon. As an alternative, PC-DARTS (Xu et al.,

2020) proposes a partial channel connection, where only a portion of channels is sent to the mixed-operation. However, partial connection can cause loss of information as shown in section 3.1.5 and PC-DARTS searches on a shallower network with less channels, suffering the search and evaluation gap. Our solution, by progressively pruning the operation space and meanwhile widening the network, searches in a task-specific manner and achieves superior accuracy on challenging datasets like ImageNet (+2.8% over BayesNAS, +2.3% over GDAS, +2.3% over PARSEC, +2.0% over DSNAS, +1.2% over ProxylessNAS, and +0.5% over PC-DARTS).

3.3 Experiments

In this section, we evaluate our proposed DrNAS on two search spaces: the CNN search space in DARTS (Liu et al., 2018b) and NAS-Bench-201 (Dong and Yang, 2020). For DARTS space, we conduct experiments on both CIFAR-10 and ImageNet in section 3.3.1 and 3.3.2 respectively. For NAS-Bench-201, we test all 3 supported datasets (CIFAR-10, CIFAR-100, ImageNet-16-120 (Chrabaszcz et al., 2017)) in section 3.3.3. Furthermore, we empirically study the dynamics of exploration and exploitation throughout the search process in section 3.3.4.

3.3.1 Results on CIFAR-10

Architecture Space For both search and evaluation phases, we stack 20 cells to compose the network and set the initial channel number as 36. We place the reduction cells at the 1/3 and 2/3 of the network and each cell consists of $N = 6$ nodes.

Search Settings We equally divide the 50K training images into two parts, one is used for optimizing the network weights by momentum SGD and the other for learning the Dirichlet architecture distribution by an Adam optimizer. Since Dirichlet concentration β must be positive, we apply the shifted exponential linear mapping $\beta = \mathrm{ELU}(\eta) + 1$ and optimize over η instead. We use l_2 norm to constrain the distance between η and the anchor $\hat{\eta} = 0$. The η is initialized by standard Gaussian with scale 0.001, and λ in (3.1) is set to 0.001. The ablation study in Appendix 3.3.7 reveals the effectiveness of our anchor regularizer, and DrNAS is insensitive to a wide range of λ. These settings are consistent for all experiments. For progressive architecture learning, the whole search process consists of 2 stages, each with 25 iterations. In the first stage, we set the partial channel parameter K as 6 to fit the super-network into a single GTX 1080Ti GPU with 11GB memory, i.e., only 1/6 features are sampled on each edge. For the second stage, we prune half candidates and meanwhile widen the network twice, i.e., the operation space size reduces from 8 to 4 and K becomes 3.

Retrain Settings The evaluation phase uses the entire 50K training set to train the network from scratch for 600 epochs. The network weight is optimized by an SGD optimizer with a cosine annealing learning rate initialized as 0.025, a momentum of 0.9, and a weight decay of 3×10^{-4}. To allow a fair comparison with previous work, we also employ cutout regularization with length 16, drop-path (Zoph et al., 2018) with probability 0.3 and an auxiliary tower of weight 0.4.

Results Table 3.2 summarizes the performance of DrNAS compared with other popular NAS methods, and we also visualize the searched cells in Appendix 3.3.5. DrNAS achieves an average test error of 2.46%, ranking top amongst recent NAS results. ProxylessNAS is the only method that achieves lower test error than us, but it searches on a different space with a much longer search time and has larger model size. We also perform experiments to assign proper credit to the two parts of our proposed algorithm, i.e., Dirichlet architecture distribution and progressive learning scheme. When searching on a proxy task with 8 stacked cells and 16 initial channels as the convention (Liu et al., 2018b; Xu et al., 2020), we achieve a test error of 2.54% that surpasses most baselines. Our progressive learning algorithm eliminates the gap between the proxy and target tasks, which further reduces the test error.

Consequently, both of the two parts contribute a lot to our performance gains.

Table 3.2: Comparison with state-of-the-art image classifiers on CIFAR-10.

Architecture	**Test Error**	**Params**	**Search Cost**	**Search**
	(%)	**(M)**	**(GPU days)**	**Method**
DenseNet-BC (Huang et al., 2017)[⋆]	3.46	25.6	-	manual
NASNet-A (Zoph et al., 2018)	2.65	3.3	2000	RL
AmoebaNet-A (Real et al., 2019a)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-B (Real et al., 2019a)	2.55 ± 0.05	2.8	3150	evolution
PNAS (Liu et al., 2018a)[⋆]	3.41 ± 0.09	3.2	225	SMBO
ENAS (Pham et al., 2018)	2.89	4.6	0.5	RL
DARTS (1st) (Liu et al., 2018b)	3.00 ± 0.14	3.3	0.4	gradient
DARTS (2nd) (Liu et al., 2018b)	2.76 ± 0.09	3.3	1.0	gradient
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	2.8	1.5	gradient
GDAS (Dong and Yang, 2019)	2.93	3.4	0.3	gradient
BayesNAS (Zhou et al., 2019)	2.81 ± 0.04	3.4	0.2	gradient
ProxylessNAS (Cai et al., 2019)[†]	2.08	5.7	4.0	gradient
PARSEC (Casale et al., 2019)	2.81 ± 0.03	3.7	1	gradient
P-DARTS (Chen et al., 2019)	2.50	3.4	0.3	gradient
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	3.6	0.1	gradient
SDARTS-ADV (Chen and Hsieh, 2020)	2.61 ± 0.02	3.3	1.3	gradient
GAEA + PC-DARTS (Li et al., 2020)	2.50 ± 0.06	3.7	0.1	gradient
DrNAS (without progressive learning)	2.54 ± 0.03	4.0	0.4[‡]	gradient
DrNAS	2.46 ± 0.03	4.1	0.6[‡]	gradient

[⋆] Obtained without cutout augmentation.

[†] Obtained on a different space with PyramidNet (Han et al., 2017) as the backbone.

[‡] Recorded on a single GTX 1080Ti GPU.

3.3.2 Results on ImageNet

Architecture Space The network architecture for ImageNet is slightly different from that for CIFAR-10 in that we stack 14 cells and set the initial channel number as 48. We also first downscale the spatial resolution from 224×224 to 28×28 with three convolution layers of stride 2 following previous works (Chen et al., 2019; Xu et al., 2020). The other settings are the same with section 3.3.1.

Search Settings Following PC-DARTS (Xu et al., 2020), we randomly sample 10% and 2.5% images from the 1.3M training set to alternatively learn network weight and Dirichlet architecture distribution by a momentum SGD and an Adam optimizer respectively. We use 8 RTX 2080 Ti GPUs for both search and evaluation, and the setup of progressive pruning is the same with that on CIFAR-10, i.e., 2 stages with operation space size shrinking from 8 to 4, and the partial channel K reduces from 6 to 3.

Retrain Settings For architecture evaluation, we train the network for 250 epochs by an SGD optimizer with a momentum of 0.9, a weight decay of 3×10^{-5}, and a linearly decayed learning rate initialized as 0.5. We also use label smoothing and an auxiliary tower of weight 0.4 during training. The learning rate warm-up is employed for the first 5 epochs following previous works (Chen et al., 2019; Xu et al., 2020).

Results As shown in Table 3.3, we achieve a top-1/5 test error of 23.7%/7.1%, outperforming all compared baselines and achieving state-of-the-art performance in the ImageNet mobile setting. The searched cells are visualized in Appendix 3.3.5. Similar to section 3.3.1, we also report the result achieved with 8 cells and 16 initial channels, which is a common setup for the proxy task on ImageNet (Xu et al., 2020). The obtained 24.2% top-1 accuracy is already highly competitive, which demonstrates the effectiveness of the architecture distribution learning on large-scale tasks. Then our progressive learning scheme further increases the top-1/5 accuracy for 0.5%/0.2%. Therefore, learning in a task-specific manner is essential to discover better architectures.

Table 3.3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	**Test Error(%)**		**Params**	**Search Cost**	**Search**
	top-1	top-5	**(M)**	**(GPU days)**	**Method**
Inception-v1 (Szegedy et al., 2015)	30.1	10.1	6.6	-	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	-	manual
ShuffleNet 2× (v1) (Zhang et al., 2018c)	26.4	10.2	~ 5	-	manual
ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	-	~ 5	-	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	2000	RL
AmoebaNet-C (Real et al., 2019a)	24.3	7.6	6.4	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	225	SMBO
MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	-	RL
DARTS (2nd) (Liu et al., 2018b)	26.7	8.7	4.7	1.0	gradient
SNAS (mild) (Xie et al., 2019)	27.3	9.2	4.3	1.5	gradient
GDAS (Dong and Yang, 2019)	26.0	8.5	5.3	0.3	gradient
BayesNAS (Zhou et al., 2019)	26.5	8.9	3.9	0.2	gradient
DSNAS (Hu et al., 2020)†	25.7	8.1	-	-	gradient
ProxylessNAS (GPU) (Cai et al., 2019)†	24.9	7.5	7.1	8.3	gradient
PARSEC (Casale et al., 2019)	26.0	8.4	5.6	1	gradient
P-DARTS (CIFAR-10) (Chen et al., 2019)	24.4	7.4	4.9	0.3	gradient
P-DARTS (CIFAR-100) (Chen et al., 2019)	24.7	7.5	5.1	0.3	gradient
PC-DARTS (CIFAR-10) (Xu et al., 2020)	25.1	7.8	5.3	0.1	gradient
PC-DARTS (ImageNet) (Xu et al., 2020)†	24.2	7.3	5.3	3.8	gradient
GAEA + PC-DARTS (Li et al., 2020)†	24.0	7.3	5.6	3.8	gradient
DrNAS (without progressive learning)†	24.2	7.3	5.2	3.9	gradient
DrNAS†	23.7	7.1	5.7	4.6	gradient

† The architecture is searched on ImageNet, otherwise it is searched on CIFAR-10 or CIFAR-100.

3.3.3 Results on NAS-Bench-201

Recently, some researchers doubt that the expert knowledge applied to the evaluation protocol plays an important role in the impressive results achieved by leading NAS methods (Li and Talwalkar, 2019; Yang et al., 2020). So to further verify the effectiveness of DrNAS, we perform experiments on NAS-Bench-201 (Dong and Yang, 2020), where architecture performance can be directly obtained by querying in the database. NAS-Bench-201 provides support for 3 dataset (CIFAR-10, CIFAR-100, ImageNet-16-120 (Chrabaszcz et al., 2017)) and has a unified cell-based search space containing 15,625 architectures. We refer to their paper (Dong and Yang, 2020) for details of the space. Our experiments are performed in a task-specific manner, i.e., the search and evaluation are based on the same dataset. The hyperparameters for all compared methods are set as their default and for DrNAS, we use the same search settings with section 3.3.1. We run every method 4 independent times with different random seeds and report the mean and standard deviation in Table 3.4.

As shown, we achieve the best accuracy on all 3 datasets. On CIFAR-100, we even achieve the global optimal. Specifically, DrNAS outperforms DARTS, GDAS, DSNAS, PC-DARTS, and SNAS by 103.8%, 35.9%, 30.4%, 6.4%, and 4.3% on average. We notice that the two methods (GDAS and DSNAS) that enforce a discrete constraint, i.e., only sample a single path every search iteration, perform undesirable especially on CIFAR-100. In comparison, SNAS, employing a similar Gumbel-softmax trick but without the discretization, performs much better. Consequently, a discrete constraint during search can reduce the GPU memory consumption but empirically suffers instability. In comparison, we develop the progressive learning scheme on top of the architecture distribution learning, enjoying both memory efficiency and strong search performance.

Table 3.4: Comparison with state-of-the-art NAS methods on NAS-Bench-201.

Method	**CIFAR-10**		**CIFAR-100**		**ImageNet-16-120**	
	validation	test	validation	test	validation	test
ResNet (He et al., 2016)	90.83	93.97	70.42	70.86	44.53	43.63
Random (baseline)	90.93 ± 0.36	93.70 ± 0.36	70.60 ± 1.37	70.65 ± 1.38	42.92 ± 2.00	42.96 ± 2.15
RSPS (Li and Talwalkar, 2019)	84.16 ± 1.69	87.66 ± 1.69	45.78 ± 6.33	46.60 ± 6.57	31.09 ± 5.65	30.78 ± 6.12
Reinforce (Zoph et al., 2018)	91.09 ± 0.37	93.85 ± 0.37	70.05 ± 1.67	70.17 ± 1.61	43.04 ± 2.18	43.16 ± 2.28
ENAS (Pham et al., 2018)	39.77 ± 0.00	54.30 ± 0.00	10.23 ± 0.12	10.62 ± 0.27	16.43 ± 0.00	16.32 ± 0.00
DARTS (1st) (Liu et al., 2018b)	39.77 ± 0.00	54.30 ± 0.00	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
DARTS (2nd) (Liu et al., 2018b)	39.77 ± 0.00	54.30 ± 0.00	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
GDAS (Dong and Yang, 2019)	90.01 ± 0.46	93.23 ± 0.23	24.05 ± 8.12	24.20 ± 8.08	40.66 ± 0.00	41.02 ± 0.00
SNAS (Xie et al., 2019)	90.10 ± 1.04	92.77 ± 0.83	69.69 ± 2.39	69.34 ± 1.98	42.84 ± 1.79	43.16 ± 2.64
DSNAS (Hu et al., 2020)	89.66 ± 0.29	93.08 ± 0.13	30.87 ± 16.40	31.01 ± 16.38	40.61 ± 0.09	41.07 ± 0.09
PC-DARTS (Xu et al., 2020)	89.96 ± 0.15	93.41 ± 0.30	67.12 ± 0.39	67.48 ± 0.89	40.83 ± 0.08	41.31 ± 0.22
DrNAS	$\mathbf{91.55 \pm 0.00}$	$\mathbf{94.36 \pm 0.00}$	$\mathbf{73.49 \pm 0.00}$	$\mathbf{73.51 \pm 0.00}$	$\mathbf{46.37 \pm 0.00}$	$\mathbf{46.34 \pm 0.00}$
optimal	91.61	94.37	73.49	73.51	46.77	47.31

3.3.4 Empirical Study on Exploration v.s. Exploitation

We further conduct an empirical study on the dynamics of exploration and exploitation in the search phase of DrNAS on NAS-Bench-201. After every search epoch, We sample 100 θs from the learned Dirichlet distribution and take the $\arg\max$ to obtain 100 discrete architectures. We then plot the range of their accuracy along with the architecture selected by Dirichlet mean (solid line in Figure 3.1). Note that in our algorithm, we simply derive the architecture according to the Dirichlet mean as described in Section 3.1.1. As shown in Figure 3.1, the accuracy range of the sampled architectures starts very wide but narrows gradually during the search phase. It indicates that DrNAS learns to encourage exploration in the search space at the early stages and then gradually reduces it towards the end as the algorithm becomes more and more confident of the current choice. Moreover, the performance of our architectures can consistently match the best performance of the sampled architectures, indicating the effectiveness of DrNAS.

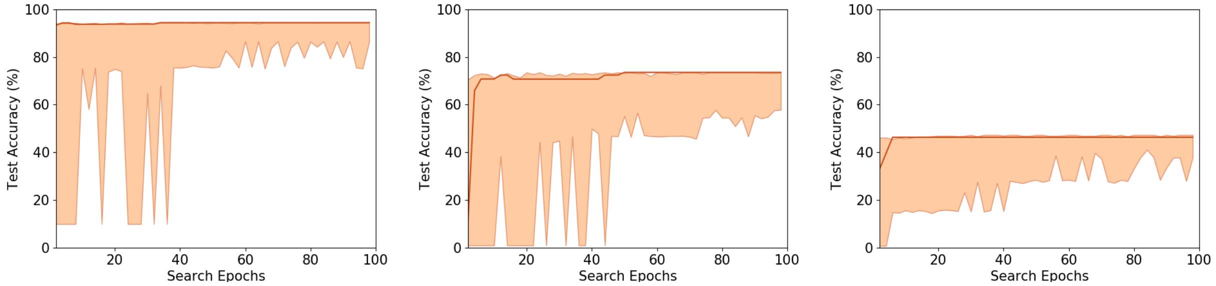


Figure 3.1: Accuracy range (min-max) of the 100 sampled architectures on CIFAR-10 (Left), CIFAR-100 (Middle), and ImageNet16-120 (Right). Note that the solid line is our derived architecture according to the Dirichlet mean as described in Section 3.1.1.

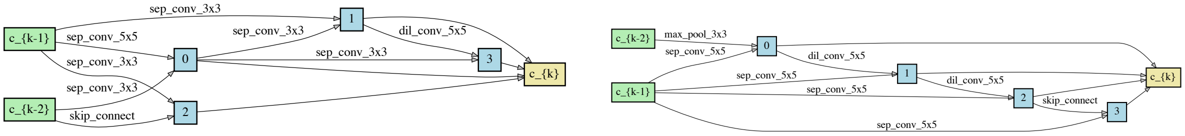


Figure 3.2: Normal (Left) and Reduction (Right) cells discovered by DrNAS on CIFAR-10.

3.3.5 Searched Architectures

We visualize the searched normal and reduction cells in Figure 3.2 and 3.3, which is directly searched on CIFAR-10 and ImageNet respectively.

3.3.6 Empirical Study on the Hessian Regularization Effect

We track the anytime Hessian norm on NAS-Bench-201 in Figure 3.4. The result is obtained by averaging from 4 independent runs. We observe that the largest eigenvalue expands about 10 times when searching by DARTS for 100 epochs. In comparison, DrNAS always maintains the Hessian norm at a low level, which is in agreement with our theoretical analysis in Section 3.1.2. Figure 3.5 shows the regularization effect under various λs. As we can see, DrNAS can keep hessian norm at a low level for a wide range of λs, which is in accordance to the relatively stable performance in Table 3.6.

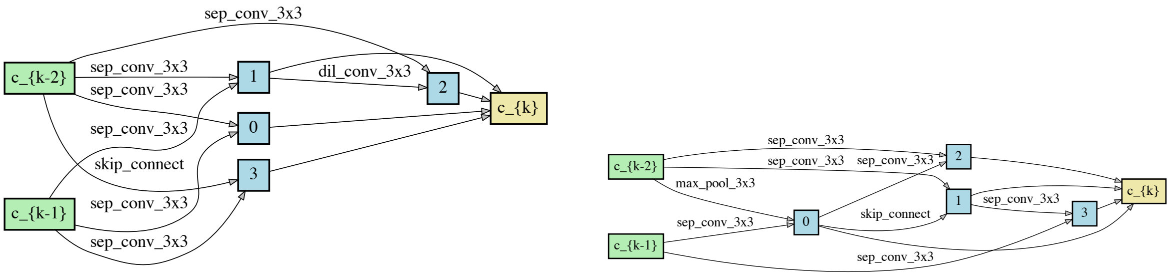


Figure 3.3: Normal (Left) and Reduction (Right) cells discovered by DrNAS on ImageNet.

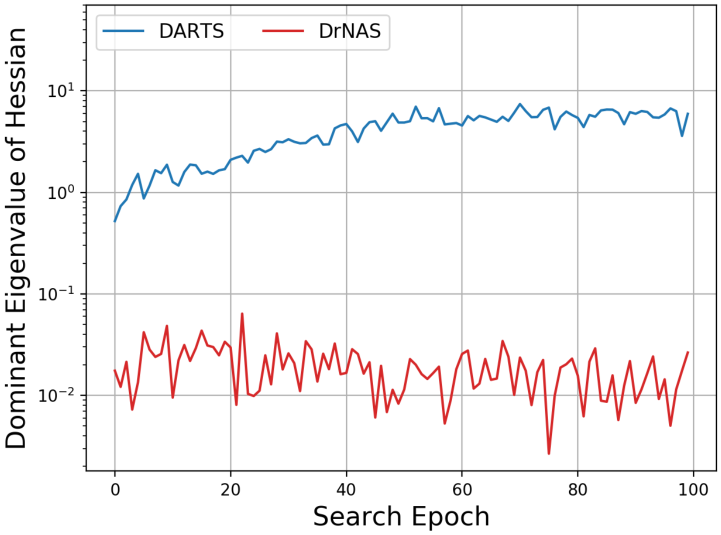


Figure 3.4: Trajectory of the Hessian norm on NAS-Bench-201 when searching with CIFAR-10 (best viewed in color).

Table 3.5: CIFAR-10 test error on 4 simplified spaces.

	s1	s2	s3	s4
DARTS	3.84	4.85	3.34	7.20
R-DARTS (DP)	3.11	3.48	2.93	3.58
R-DARTS (L2)	2.78	3.31	2.51	3.56
DrNAS (ours)	**2.74**	**2.47**	**2.4**	**2.59**

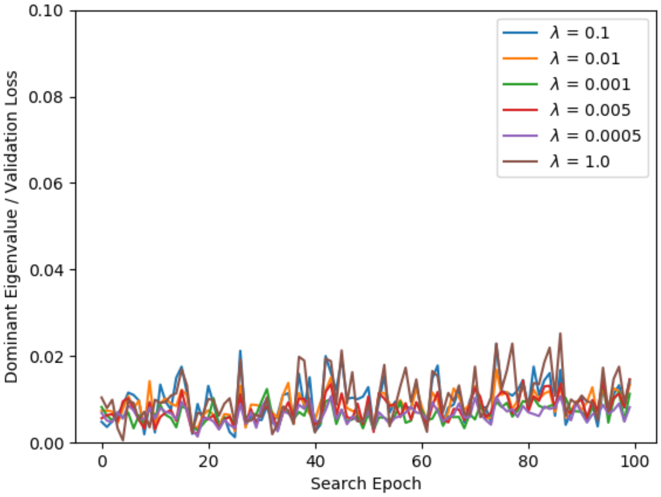


Figure 3.5: Trajectory of the Hessian norm under various λs on NAS-Bench-201 when searching with CIFAR-10 (best viewed in color).

Moreover, we compare DrNAS with DARTS and R-DARTS on 4 simplified space proposed in (Zela et al., 2020b) and record the endpoint dominant eigenvalue. The first space S1 contains 2 popular operators per edge based on DARTS search result. For S2, S3, and S4, the operation sets are $\{3 \times 3$ *separable convolution*, *skip connection*$\}$, $\{3 \times 3$ *separable convolution*, *skip connection*, *zero*$\}$, and $\{3 \times 3$ *separable convolution*, *noise*$\}$ respectively. As shown in Table 3.5, DrNAS consistently outperforms DARTS and R-DARTS. The endpoint eigenvalues for DrNAS are 0.0392, 0.0390, 0.0286, 0.0389 respectively. Figure 3.5 shows the Hessian norm trajectory under various λ.

3.3.7 Ablation Study on Anchor Regularizer Parameter

Table 3.6 shows the accuracy of the searched architecture using different value of λ while keeping all other settings the same. Using anchor regularizer? for a wide range of value can boost the accuracy and DrNAS performs quite stable under different λs.

Table 3.6: Test accuracy of the searched architecture with different λs on NAS-Bench-201 (CIFAR-10). $\lambda = 1e^{-3}$ is what we used for all of our experiments.

λ	0	$5e^{-4}$	$1e^{-3}$	$5e^{-3}$	$1e^{-2}$	$1e^{-1}$	1
Accuracy	93.78	94.01	94.36	94.36	94.36	93.76	93.76

CHAPTER 4

Neural Architecture Search in Collaborative Filtering

Collaborative filtering (CF) (Herlocker et al., 1999; Su and Khoshgoftaar, 2009) is an important topic in both machine learning and data mining. By capturing interactions among rows and columns in a data matrix, CF predicts the missing entries based on the observed elements. The most famous CF application is the recommender system (Koren, 2008). The ratings in such systems can be arranged as a data matrix, where rows correspond to users, columns are the items, and the entries are ratings collected. Since users usually only interact with a few items, there will be lots of missing entries in the rating matrix. The task is to estimate users' ratings on items they have not yet explored. Due to the good empirical performance of CF approaches, they also have been used in various other applications as well, e.g., image in-painting in computer vision (Ji et al., 2010), link prediction in social networks (Kim and Leskovec, 2011) and topic modeling for text analysis Wang and Blei (2011). More recently, CF is also extended to tensor data (i.e., higher-order matrices) Kolda and Bader (2009) to handle side-information, e.g., extra features Karatzoglou et al. (2010) and time (Lei et al., 2009).

In the last decade, low-rank matrix factorization (Koren, 2008; Mnih and Salakhutdinov, 2008) has been the most popular approach to CF. It can be formulated as the following optimization problem:

$$\min_{\boldsymbol{U},\boldsymbol{V}} \sum\nolimits_{(i,j)\in\Omega} \ell\left(\boldsymbol{u}_i^\top \boldsymbol{v}_j, \boldsymbol{O}_{ij}\right) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2, \tag{4.1}$$

where ℓ is a loss function, the observed elements are indicated by Ω with values given by the corresponding positions in matrix $\boldsymbol{O} \in \mathbb{R}^{m\times n}$, $\lambda \geq 0$ is a hyper-parameter, and $\boldsymbol{u}_i \in \mathbb{R}^k$ and

$\boldsymbol{v}_j \in \mathbb{R}^k$ are embedded vectors for user i and item j, respectively. Note that (4.1) captures interactions between user $\boldsymbol{u}_i$ and item $\boldsymbol{v}_j$ by the inner product. This achieves good empirical performance, enjoys sound statistical guarantees (Candès and Recht, 2009; Recht et al., 2010) (e.g., the data matrix can be exactly recovered when $\boldsymbol{O}$ satisfies certain incoherent conditions and the missing entries follow some distributions), and fast training (Gemulla et al., 2011; Mnih and Salakhutdinov, 2008) (e.g., can be trained end-to-end by stochastic optimization).

While the inner product has many benefits, it may not yield the best performance for various CF tasks due to the complex nature of user-item interactions. For example, the plus operation, which quantifies the users' preference based on $\boldsymbol{u}_i - \boldsymbol{v}_j$, has been explored in (He et al., 2017; Hsieh et al., 2017). The motivation is that the embedded vectors obtained from inner product do not satisfy the triangle inequality, while those from the plus operation can. Other operations (such as concatenation and convolution) have also outperformed the inner product on many CF tasks (He et al., 2018; Kim et al., 2016; Rendle, 2012). Due to the success of deep networks (Goodfellow et al., 2016), the multi-layer perceptron (MLP) is also recently used as the interaction function (IFC) in CF (Cheng et al., 2016; He et al., 2017; Xue et al., 2017), and achieves good performance. However, choosing and designing an IFC is not easy, IFC should also depend on the data sets and tasks. Using one simple operation, such as the inner product or plus, may not be expressive enough to ensure good performance. On the other hand, directly using a MLP leads to the difficult and time-consuming task of architecture selection (Baker et al., 2017; Zhang et al., 2019; Zoph and Le, 2017). Thus, it is hard to have an objectively best IFC across different tasks and data sets (Dacrema et al., 2019).

Recently, there have been a lot of interests in automated machine learning (AutoML) (Hutter et al., 2018), which involves searching for an appropriate network architecture by reinforcement learning (Baker et al., 2017; Zoph et al., 2017; Zoph and Le, 2017), and the fine-tuning of a classifier ensemble by Bayes optimization (Feurer et al., 2015). In this chapter, motivated by the success of AutoML, we consider formulating the search for interaction

Table 4.1: Popular human-designed interaction functions (IFC) for CF, where $\boldsymbol{H}$ is a parameter to be trained. SIF searches a proper IFC from the validation set (i.e., by AutoML), while others are all designed by experts.

	IFC	operation	space	predict time	recent examples
human-designed	$\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$	inner product	$O((m+n)k)$	$O(k)$	MF (Koren, 2008), FM (Rendle, 2012)
	$\boldsymbol{u}_i - \boldsymbol{v}_j$	plus (minus)	$O((m+n)k)$	$O(k)$	CML (Hsieh et al., 2017)
	$\max(\boldsymbol{u}_i, \boldsymbol{v}_j)$	max, min	$O((m+n)k)$	$O(k)$	ConvMF (Kim et al., 2016)
	$\sigma([\boldsymbol{u}_i; \boldsymbol{v}_j])$	concat	$O((m+n)k)$	$O(k)$	Deep&Wide (Cheng et al., 2016)
	$\sigma(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}[\boldsymbol{u}_i; \boldsymbol{v}_j])$	multi, concat	$O((m+n)k)$	$O(k^2)$	NCF (He et al., 2017)
	$\boldsymbol{u}_i * \boldsymbol{v}_j$	conv	$O((m+n)k)$	$O(k \log(k))$	ConvMF (Kim et al., 2016)
	$\boldsymbol{u}_i \otimes \boldsymbol{v}_j$	outer product	$O((m+n)k)$	$O(k^2)$	ConvNCF (He et al., 2018)
AutoML	SIF (proposed)	searched	$O((m+n)k)$	$O(k)$	——

functions (SIF) as an AutoML problem. Inspired by observations on existing IFCs, we first generalize the CF objective and define the SIF problem. These observations also help to identify a domain-specific and expressive search space, which not only includes many human-designed IFCs, but also covers new ones not yet explored in the literature. We further represent the SIF problem, armed with the designed search space, as a structured MLP. This enables us to derive an efficient search algorithm based on one-shot architecture search (Liu et al., 2018b; Xie et al., 2018; Yao et al., 2019). The algorithm can jointly train the embedding vectors and search IFCs in a stochastic end-to-end manner. We further extend the proposed SIF, including both the search space and one-shot search algorithm, to handle tensor data. Finally, we perform experiments on CF tasks with both matrix data (i.e., MovieLens data sets) and tensor data (i.e., Youtube data set).

Notations

Vectors are denoted by lowercase boldface, and matrices by uppercase boldface. For two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product, $\boldsymbol{x} \odot \boldsymbol{y}$ is the element-wise product, $\boldsymbol{x} \otimes \boldsymbol{y}$ is the outer product, $[\boldsymbol{x}; \boldsymbol{y}]$ concatenates (denoted "concat") two vectors to a longer one, and $\boldsymbol{x} * \boldsymbol{y}$

is the convolution (denoted "conv"). $\text{Tr}(\boldsymbol{X})$ is the trace of a square matrix $\boldsymbol{X}$, and $\|\boldsymbol{X}\|_F$ is the Frobenius norm. $\|\boldsymbol{x}\|_2$ is the ℓ_2-norm of a vector $\boldsymbol{x}$, and $\|\boldsymbol{x}\|_0$ counts its number of nonzero elements.

4.1 Existing Interaction Functions (IFCs)

As mentioned above, The IFC is key to CF, with the inner product being the most popular operation (Candès and Recht, 2009; Koren, 2008; Mnih and Salakhutdinov, 2008). However, due to the complex interactions among users and items, many CF models other than low-rank matrix factorization have been proposed. Examples include the factorization machine (FM) (Rendle, 2012), collaborative metric learning (CML) (Hsieh et al., 2017), convolutional matrix factorization (ConvMF) (Kim et al., 2016), Deep & Wide (Cheng et al., 2016), neural collaborative filtering (NCF) (He et al., 2017), and convolutional neural collaborative filtering (ConvNCF) (He et al., 2018). These models are summarized in Table 4.1, As can be seen, many operations other than the simple inner product have been used, and have achieved better performance than matrix factorization on many CF tasks. Moreover, they all have the same space complexity, which grows linearly w.r.t. m, n and k, but with different time complexities. While the design of IFCs is very important, this depends highly on the given data and task, and there is no single model in Table 4.1 that consistently outperforms the rest across all CF tasks (Aggarwal, 2017; Su and Khoshgoftaar, 2009). Thus, it is of great importance to select a proper IFC from a set of customized IFC's designed by humans, or to design a new IFC which has not been visited in the literature.

4.2 Proposed Method

In Section 4.1, we have witnessed the importance of IFCs, and the difficulty of choosing or designing one for the given task and data. Similar observations have also been made in designing neural networks, which motivates NAS methods for deep networks (Baker et al., 2017; Zoph and Le, 2017). Moreover, NAS has been developed as a replacement of humans, which can discover data- and task-dependent architectures with better performance. These inspire us to search for proper IFCs in CF by AutoML approaches.

4.2.1 Problem Definition

First, we define the AutoML problem here and identify an expressive search space for IFCs, which includes the various operations in Table 4.1. Inspired by generalized matrix factorization (He et al., 2017; Xue et al., 2017) and objective (4.1), we propose the following generalized CF objective:

$$\begin{aligned} \min F(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}) \equiv & \sum\nolimits_{(i,j)\in\Omega} \ell\left(\boldsymbol{w}^\top f\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right), \boldsymbol{O}_{ij}\right) \\ & + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2, \text{ s.t. } \|\boldsymbol{w}\|_2 \leq 1, \end{aligned} \tag{4.2}$$

where f is the IFC (which takes the user embedding vector $\boldsymbol{u}_i$ and item embedding vector $\boldsymbol{v}_j$ as input, and outputs a vector), and $\boldsymbol{w}$ is a learning parameter. Obviously, all the IFCs in Table 4.1 can be represented by using different f's. The following Proposition shows that the constraint $\|\boldsymbol{w}\|_2 \leq 1$ is necessary to ensure existence of a solution.

Proposition 4.2.1 *If f is an operation shown in Table 4.1 and the ℓ_2 constraint on $\boldsymbol{w}$ is removed, then F in (4.2) has no nonzero optimal solution when $\lambda > 0$.*

Proofs of Proposition 4.2.1 Taking f as the inner product function as an example. Let $\boldsymbol{A} = \{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}\} \neq \boldsymbol{0}$ be an optimal point of F, then

$$F(\boldsymbol{A}) = \sum\nolimits_{(i,j)\in\Omega} \ell\left(\boldsymbol{w}^\top f\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right), \boldsymbol{O}_{ij}\right)^2 + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2. \tag{4.3}$$

We construct another $\boldsymbol{A}' = \{\beta\boldsymbol{U}, \beta\boldsymbol{V}, \frac{1}{\beta^2}\boldsymbol{w}\}$ with $\beta \in (0, 1)$, then

$$F\left(\boldsymbol{A}'\right) = \sum_{(i,j)\in\Omega} \ell\left(\boldsymbol{w}^\top f\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right), \boldsymbol{O}_{ij}\right)^2 + \frac{\lambda\beta^2}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda\beta^2}{2}\|\boldsymbol{V}\|_F^2 < F(\boldsymbol{A}),$$

which violates the assumption that $\boldsymbol{A} \neq \boldsymbol{0}$ is an optimal solution. The same holds for f being other operations in Table4.1. Thus the proposition holds.

Based on above objective, we now define the AutoML problem, i.e., searching interaction functions (SIF) for CF, here.

Definition 4.2.1 (SIF problem) *Let $\mathcal{M}$ be a performance measure (the lower the better) defined on the validation set $\bar{\Omega}$ (disjoint from Ω), and $\mathcal{F}$ be a family of vector-valued functions with two vector inputs. The problem of searching for an interaction function (SIF), i.e., finding f^*, is defined as*

$$f^* = \arg\min_{f \in \mathcal{F}} \sum\nolimits_{(i,j) \in \bar{\Omega}} \mathcal{M}\left(f(\boldsymbol{u}_i^*, \boldsymbol{v}_j^*)^\top \boldsymbol{w}^*, \boldsymbol{O}_{ij}\right) \tag{4.4}$$
$$s.t.\ [\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{w}^*] = \arg\min_{\boldsymbol{U},\boldsymbol{V},\boldsymbol{w}} F(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}),$$

where $\boldsymbol{u}_i^*$ (resp. $\boldsymbol{v}_j^*$) is the ith column of $\boldsymbol{U}^*$ (resp. jth column of $\boldsymbol{V}^*$).

Similar to other AutoML problems (such as auto-sklearn (Feurer et al., 2015) and NAS (Baker et al., 2017; Zoph and Le, 2017)), SIF is a bi-level optimization problem (Colson et al., 2007). On the top level, a good architecture f is searched based on the validation set. On the lower level, we find the model parameters using F on the training set. Due to the nature of bi-level optimization, AutoML problems are difficult to solve in general. In the following, we show how to design an expressive search space (Section 4.2.2), propose an efficient and one-shot search algorithm (Section 4.2.3), and extend the proposed method to tensor data (Section 4.2.4).

4.2.2 Designing a Search Space

Because of the powerful approximation capability of deep networks (Raghu et al., 2017), NCF (He et al., 2017) and Deep&Wide (Cheng et al., 2016) use a MLP as f. SIF then becomes searching a suitable MLP from the family $\mathcal{F}$ based on the validation set, where both MLP's architectures and weights can be searched. However, a direct search of this MLP can be expensive and difficult, since determining its architecture is already an extremely time-consuming problem as observed in the NAS literature (Liu et al., 2018b; Luo et al., 2018a; Zoph et al., 2017). Thus, it is preferable to use a simple but expressive search space that exploits domain-specific knowledge from experts. Notice that Table 4.1 contains operations that are:

- *element-wise*: a possibly nonlinear function operating on individual elements, i.e., the nonlinear function σ; and
- *vector-wise*: operators that operate on the whole input vector, e.g., binary operators like minus and multiplication.

Inspired by previous attempts that divide the NAS search space into micro and macro level (Liu et al., 2018b; Zoph et al., 2017), we propose to first search for a nonlinear transform on each single element, and then combine these element-wise operations on the vector level. Let $\mathcal{O}$ be an operator selected from $\{$ multi, plus, min, max, $\mathsf{concat}\}$, $g(\beta; \boldsymbol{x}) \in \mathbb{R}$ be a simple nonlinear function with input $\beta \in \mathbb{R}$ and hyper-parameter $\boldsymbol{x}$. We construct a search space $\mathcal{F}$ for (4.4), where each f is expressed as:

$$f(\boldsymbol{u}_i, \boldsymbol{v}_j) = \mathcal{O}(\dot{\boldsymbol{u}}_i, \dot{\boldsymbol{v}}_j) \tag{4.5}$$

with

$$[\dot{\boldsymbol{u}}_i]_l = g\left([\boldsymbol{u}_i]_l\, ; \boldsymbol{p}\right), \quad \text{and} \quad [\dot{\boldsymbol{v}}_j]_l = g\left([\boldsymbol{v}_j]_l\, ; \boldsymbol{q}\right),$$

where $[\boldsymbol{u}_i]_l$ (resp. $[\boldsymbol{v}_j]_l$) is the lth element of $\boldsymbol{u}_i$ (resp. $[\boldsymbol{v}_j]_l$), and $\boldsymbol{p}$ (resp. $\boldsymbol{q}$) is the hyper-parameter of g transforming user (resp. item) embeddings.

Note that we omit the convolution and outer product from $\mathcal{O}$ (vector-wise operations) in (4.5), as they need significantly more computational time and have inferior performance than the rest (see Section 4.3.4). Besides, we parameterize g with a very small MLP with fixed architecture (single input, single output and five sigmoid hidden units) for the element-wise level in (4.5), and the ℓ_2-norms of the weights, i.e., $\boldsymbol{p}$ and $\boldsymbol{q}$ in (4.5), are constrained to be ≤ 1.

Remark 4.2.1 *This search space $\mathcal{F}$ meets the requirements for AutoML. First, as it involves an extra nonlinear transformation, it contains operations that are more general than those designed by experts in Table 4.1. Such expressiveness leads to better performance than human designed models in the experiments (Section 4.3.2). Second, the search space is much more constrained than that of a general MLP mentioned above, as we only need to select an operation for $\mathcal{O}$ and determine the weights for a small fixed MLP.*

4.2.3 Efficient One-Shot Search Algorithm

Usually, AutoML problems are expensive to search, as full model training is required. In this section, we propose an efficient algorithm, which only approximately trains the models, to search the space in an end-to-end and stochastic manner. Our algorithm is motivated by the recent success of one-shot architecture search.

Continuous representation of the space Note that the search space introduced by (4.5) contains both discrete (i.e., choice of operations) and continuous variables (i.e., hyper-parameter $\boldsymbol{p}$ and $\boldsymbol{q}$ for nonlinear transformation), which is generally inefficient to conduct search. Motivated by differentiable search in NAS (Liu et al., 2018b; Xie et al., 2018), we propose to relax the choices among operations as a sparse vector in a continuous space. Specifically, we transform f in (4.5) as

$$h_\alpha(\boldsymbol{u}_i, \boldsymbol{v}_j) \equiv \sum\nolimits_{m=1}^{|\mathcal{O}|} \alpha_m \left(\boldsymbol{w}_m^\top \mathcal{O}_m(\dot{\boldsymbol{u}}_i, \dot{\boldsymbol{v}}_j) \right) \quad \text{s.t.} \quad \boldsymbol{\alpha} \in \mathcal{C}, \tag{4.6}$$

where $\boldsymbol{\alpha} = [\alpha_m]$ and $\mathcal{C}$ enforces that only one operation is selected. Since operations may lead to different output sizes, we associate each operation m with its own $\boldsymbol{w}_m$.

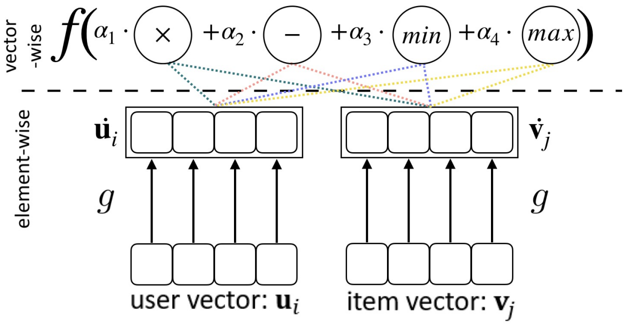

vector -wise
$f(\alpha_1 \cdot (\times) + \alpha_2 \cdot (-) + \alpha_3 \cdot (min) + \alpha_4 \cdot (max))$
element-wise
$\dot{\mathbf{u}}_i$
$\dot{\mathbf{v}}_j$
g
g
user vector: $\mathbf{u}_i$
item vector: $\mathbf{v}_j$

Figure 4.1: Representing the search space as a structured MLP. Vector-wise: standard linear algebra operations; element-wise: simple non-linear transformation.

Let $\boldsymbol{T} = \{\boldsymbol{U}, \boldsymbol{V}, \{\boldsymbol{w}_m\}\}$ be the parameters to be determined by the training dataset, and $\boldsymbol{S} = \{\boldsymbol{\alpha}, \boldsymbol{p}, \boldsymbol{q}\}$ be the hyper-parameters to be determined by the validation set. Combining h_α with (4.4), we propose the following objective:

$$\min_{\boldsymbol{S}} H(\boldsymbol{S}, \boldsymbol{T}) \equiv \sum\nolimits_{(i,j)\in\bar{\Omega}} \mathcal{M}(h_\alpha(\boldsymbol{u}_i^*, \boldsymbol{v}_j^*)^\top \boldsymbol{w}_\alpha^*, \boldsymbol{O}_{ij}) \tag{4.7}$$
$$\text{s.t. } \boldsymbol{\alpha} \in \mathcal{C} \text{ and } \boldsymbol{T}^* = \arg\min_{\boldsymbol{T}} F_\alpha(\boldsymbol{T}; \boldsymbol{S}),$$

where $\boldsymbol{T}^* = \{\boldsymbol{U}^*, \boldsymbol{V}^*, \{\boldsymbol{w}_m^*\}\}$ and the training objective F_α is

$$F_\alpha(\boldsymbol{T}; \boldsymbol{S}) \equiv \sum\nolimits_{(i,j)\in\Omega} \ell(h_\alpha(\boldsymbol{u}_i, \boldsymbol{v}_j), \boldsymbol{O}_{ij}) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2,$$
$$\text{s.t. } \|\boldsymbol{w}_m\|_2 \leq 1 \text{ for } m = 1, \cdots, |\mathcal{O}|.$$

Moreover, objective (4.7) can be expressed as a structured MLP (Figure 4.1). Compared with the general MLP mentioned in Section 4.2.2, the architecture of this structured MLP is fixed and its total number of parameters is very small. After solving (4.7), we keep $\boldsymbol{p}$ and $\boldsymbol{q}$ for element-wise non-linear transformation, and pick the operation which is indicated by the only one nonzero position in the vector $\boldsymbol{\alpha}$ for vector-wise interaction. Then, we re-train the model to obtain the final user- and item-embedded vectors, i.e., $\boldsymbol{U}$, $\boldsymbol{V}$ and corresponding $\boldsymbol{w}$ in (4.2).

Optimization by one-shot architecture search Here, we present a stochastic algorithm (Algorithm 2) to optimize the structured MLP in Figure 4.1. Overall, Algorithm 2 is inspired by NASP (Yao et al., 2020b), where the relaxation of operations is defined in (4.6). Again, we need to keep a discrete representation of the architecture, i.e., $\bar{\boldsymbol{\alpha}}$ at steps 3 and 8, but optimize a continuous architecture, i.e., $\boldsymbol{\alpha}$ at step 5. The difference is that we have extra continuous hyper-parameters $\boldsymbol{p}$ and $\boldsymbol{q}$ for element-wise nonlinear transformation here. However, they can still be updated by proximal steps (see step 6), where the closed-form solution is given by $\text{prox}_{\|\cdot\|_2 \le 1}(\boldsymbol{z}) = {}^{\boldsymbol{z}}/_{\|\boldsymbol{z}\|_2}$ Parikh and Boyd (2013).

Algorithm 2 Searching Interaction Function (SIF) algorithm.

```
1: Search space $\mathcal{F}$ represented by a structured MLP (Figure 4.1);
2: for epoch $t = 1, \cdots, T$ do
3:     Select one operation $\bar{\boldsymbol{\alpha}} = \text{prox}_{\mathcal{C}_1}(\boldsymbol{\alpha})$;
4:     sample a mini-batch on validation data set;
5:     Update continuous $\boldsymbol{\alpha}$ for vector-wise operations
           $$\boldsymbol{\alpha} = \text{prox}_{\mathcal{C}_2}\left(\boldsymbol{\alpha} - \eta \nabla_{\bar{\boldsymbol{\alpha}}} H(\boldsymbol{T}, \boldsymbol{S})\right);$$
6:     Update element-wise transformation
           $$\boldsymbol{p} = \text{prox}_{\|\cdot\|_2 \le 1}\left(\boldsymbol{p} - \eta \nabla_{\boldsymbol{p}} H(\boldsymbol{T}, \boldsymbol{S})\right),$$
           $$\boldsymbol{q} = \text{prox}_{\|\cdot\|_2 \le 1}\left(\boldsymbol{q} - \eta \nabla_{\boldsymbol{q}} H(\boldsymbol{T}, \boldsymbol{S})\right);$$
7:     sample a mini-batch on training data set;
8:     Get selected operation $\bar{\boldsymbol{\alpha}} = \text{prox}_{\mathcal{C}_1}(\boldsymbol{\alpha})$;
9:     Update training parameters $\boldsymbol{T}$ with gradients on $F_{\alpha}$;
10: end for
```

4.2.4 Handling Tensor Data

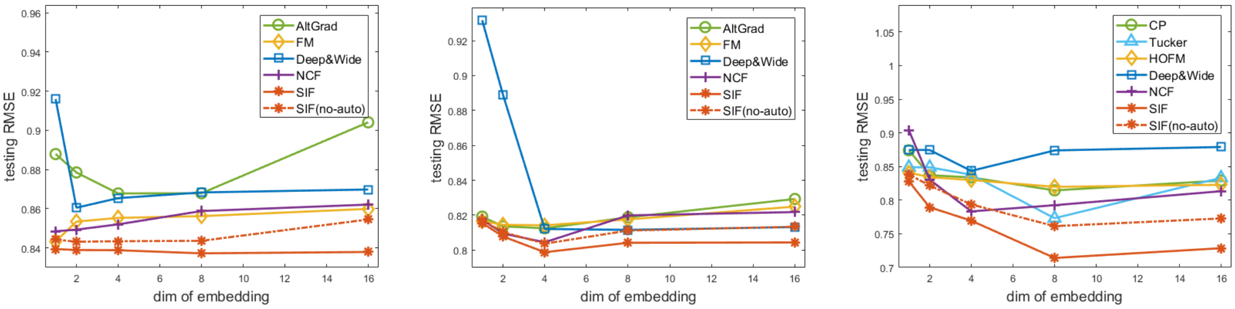
CF methods have also been used to handle tensor data. For example, low-rank matrix factorization is extended to tensor factorization, where two types of decomposition formats, i.e., CP and Tucker (Kolda and Bader, 2009), are popularly used. These two methods are also based on the inner product. Besides, the factorization machine (Rendle, 2012) is also recently extended to handle data cube (Blondel et al., 2016). These motivate us to extend the proposed SIF algorithm for tensor data. In the sequel, we focus on the 3-order tensor. Higher-order tensors can be handled in a similar way.

For tensors, we need to maintain three embedded vectors, $\boldsymbol{u}_i$, $\boldsymbol{v}_j$ and $\boldsymbol{s}_l$. First, we modify f to take three vectors as input and output another vector, and each candidate in search space (4.5) subsequently becomes $f = \mathcal{O}(\dot{\boldsymbol{u}}_i, \dot{\boldsymbol{v}}_j, \dot{\boldsymbol{s}}_l)$, where $\dot{\boldsymbol{u}}_i$'s are obtained from element-wise MLP from $\boldsymbol{u}_i$ (and similarly for $\dot{\boldsymbol{v}}_j$ and $\dot{\boldsymbol{s}}_l$). However, $\mathcal{O}$ is no longer a single operation, as three vectors are involved. $\mathcal{O}$ enumerates all possible combinations from basic operations in the matrix case. For example, if only max and $\odot$ are allowed, then $\mathcal{O}$ contains $\max(\boldsymbol{u}_i, \boldsymbol{v}_j) \odot \boldsymbol{s}_l$, $\max(\max(\boldsymbol{u}_i, \boldsymbol{v}_j), \boldsymbol{s}_l)$, $\boldsymbol{u}_i \odot \max(\boldsymbol{v}_j, \boldsymbol{s}_l)$ and $\boldsymbol{u}_i \odot \boldsymbol{v}_j \odot \boldsymbol{s}_l$. With the above modifications, it is easy to see that the space can still be represented by a structured MLP similar to that in Figure 4.1. Moreover, the proposed Algorithm 2 can still be applied. Note that the search space is much larger for tensor than matrix.

4.3 Empirical Study

4.3.1 Experiments Setup

MovieLens (matrix data) and Youtube (tensor data) are used (Table 4.2). These are benchmark data sets popularly used in the literature (Gemulla et al., 2011; Lei et al., 2009; Mnih and Salakhutdinov, 2008). Following (Wang et al., 2015; Yao and Kwok, 2018), we uniformly and randomly select 50% of the ratings for training, 25% for validation and the rest

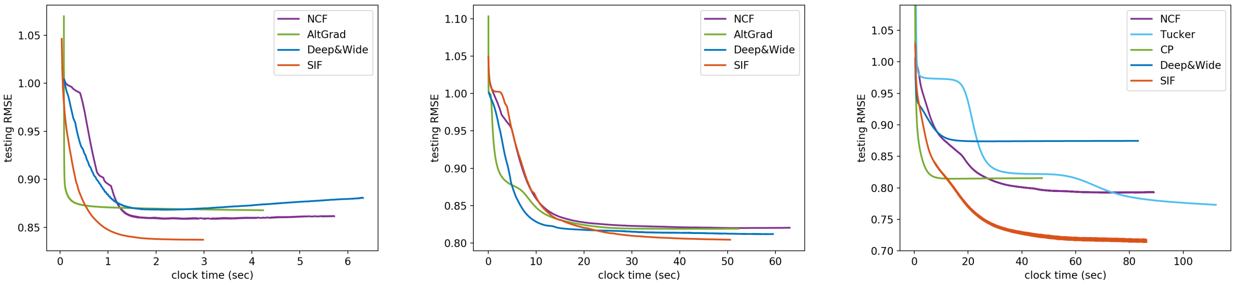


(a) MovieLens-100K.

(b) MovieLens-1M.

(c) Youtube.

Figure 4.2: Comparison of testing RMSEs between *SIF* and other CF approaches with different embedding dimension.



(a) MovieLens-100K.

(b) MovieLens-1M.

(c) Youtube.

Figure 4.3: Comparison of the convergence between *SIF* (with searched IFC) and other CF methods when embedded dimension is 8. *FM* and *HOFM* are not shown as their code donot support a callback to record testing performance.

Table 4.2: Statistic of data sets used in experiments.

data set (matrix)		users	items	ratings
MovieLens	100K	943	1,682	100,000
	1M	6,040	3,706	1,000,209

data set (tensor)	rows	columns	depths	non-zeros
Youtube	600	14,340	5	1,076,946

for testing. Note that since the size of the original Youtube dataset Lei et al. (2009) is very large (approximate 27 times the size of MovieLens-1M), we sample a subset of it to test the performance (approximately the size of MovieLens-1M). We sample rows with interactions larger than 20.

The task is to predict missing ratings given the training data set. We use the squared loss for both $\mathcal{M}$ and ℓ. For performance evaluation, we use (i) the testing RMSE as in (Gemulla et al., 2011; Mnih and Salakhutdinov, 2008): RMSE $= [{}^{1}/_{|\tilde{\Omega}|} \sum_{(i,j)\in\tilde{\Omega}} (\boldsymbol{w}^\top f(\boldsymbol{u}_i, \boldsymbol{v}_j) - \boldsymbol{O}_{ij})^2]^{1/2}$, where f is the operation chosen by the algorithm, and $\boldsymbol{w}$, $\boldsymbol{u}_i$'s and $\boldsymbol{v}_j$'s are parameters learned from the training set; and (ii) clock time (in seconds) as in (Baker et al., 2017; Liu et al., 2018b). Except for IFCs, other hyper-parameters are all tuned with grid search on the validation set. Specifically, for all *CF approaches*, we can tune the learning rate lr and the regularization coefficient λ to get the best RMSE since the network architecture is already pre-defined. We use the Adagrad Duchi et al. (2010) optimizer to perform gradient-based updates and due to its robustness, lr is not sensitive in our experiments. So we empirically fix a small learning rate lr in all our experiments. Furthermore, we utilize grid search to get the most suitable λ for all experiments, where the grid is set as $[0, 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}, 10^{-4}]$. When it comes to *AutoML approaches*, we use exactly the same lr to search for the architecture and tune λ using the same grid after the searched architecture is generated.

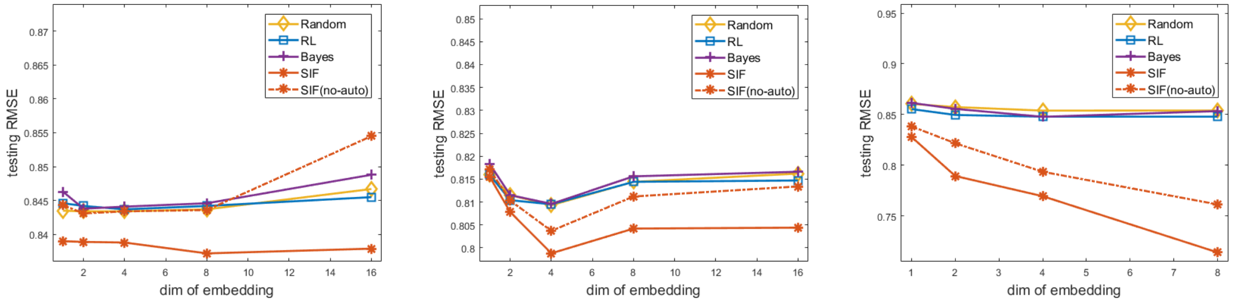
4.3.2 Comparison with State-of-the-Art CF Approaches

Here we compare SIF with popular CF approaches.

- For matrix data: The following methods for matrix data are compared: (i) alternative gradient descent ("*AltGrad*") (Koren, 2008): This is the most popular CF method, which is based on matrix factorization (i.e., inner product operation). We use gradient descent for optimization; (ii) factorization machine ("*FM*") (Rendle, 2012): This extends linear regression with matrix factorization to capture second-order interactions among features; (iii) *Deep&Wide* (Cheng et al., 2016): This is a recent CF method, which first embeds discrete features and then concatenates them for prediction; (iv) Neural collaborative filtering ("*NCF*") (He et al., 2017): This is another recent CF method, which models the IFC by neural networks.
- For tensor data: *Deep&Wide* and *NCF* can be easily extended to handle tensor data. Since the rank for the tensor is not uniquely defined, Thus, the following CF methods for tensor data are considered: Two types of popularly used low-rank factorization of tensor are used, i.e., "*CP*" and "*Tucker*" (Kolda and Bader, 2009), and gradient descent is used for optimization; "*HOFM*" (Blondel et al., 2016)": a fast variant of FM, which can capture high-order interactions.

Besides, a variant of SIF (Algorithm 2) is also compared, in which the parameter $\boldsymbol{S}$ for IFCs are optimized with training data (denoted "*SIF(no-auto)*").

Effectiveness Comparison on the testing RMSEs is shown in Figure 4.2. First, as the embedding dimension gets larger, all methods gradually overfit and the testing RMSEs get higher. *SIF(no-auto)* is slightly better than the other CF approaches, which demonstrates the expressiveness of the designed search space. Finally, with the searched IFCs, *SIF* consistently obtains lower testing RMSEs than all other CF approaches.

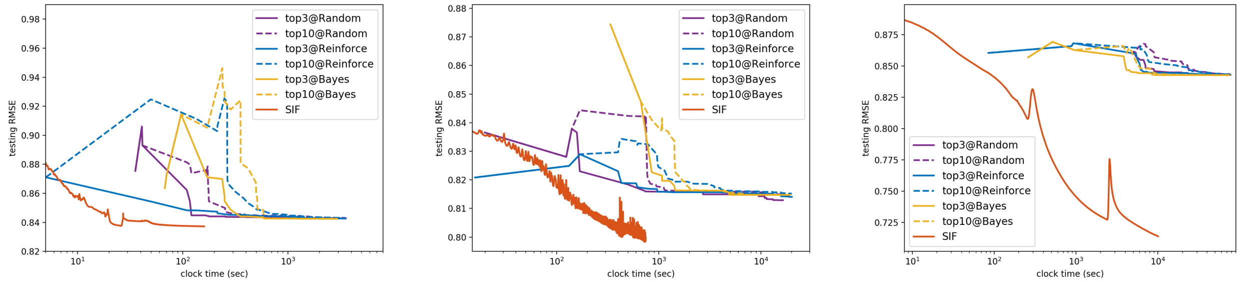


(a) MovieLens-100K.

(b) MovieLens-1M.

(c) Youtube.

Figure 4.4: Comparison of testing RMSEs between *SIF* and other AutoML approaches with different embedding dimensions. *Gen-approx* is slow with bad performance, thus is not run on Youtube.



(a) MovieLens-100K.

(b) MovieLens-1M.

(c) Youtube.

Figure 4.5: Comparison of search efficiency among *SIF* and other AutoML approaches when embedded dimension is 8.

Training efficiency If an IFC is better than another one on capturing interactions among users' and items' embeddings, it can converge faster on testing performance. Thus, we show the training efficiency with the searched interactions vs human-designed CF methods in Figure 4.3. As can be seen, the searched IFC can be more efficiently trained, which further shows the superiority of searching IFCs from the data.

Table 4.3: Total time (in seconds) comparison between SIF and CF approaches with embedding dimension being 8.

		AltGrad	FM	Deep&Wide	NCF	SIF	SIF(no-auto)
MovieLens	100K	25.4	43.1	37.9	34.3	159.8	73.4
	1M	313.7	324.3	357.0	374.9	745.3	348.7

	CP	Tucker	HOFM	Deep&Wide	NCF	SIF	SIF(no-auto)
Youtube	389.8	572.6	428.2	499.0	534.3	987.1	434.9

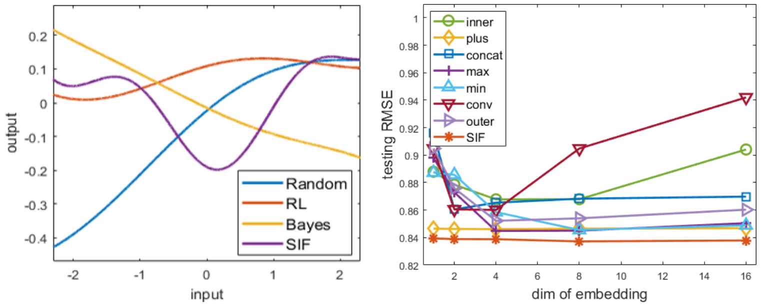
4.3.3 Comparison with State-of-the-Art AutoML Approaches

The following popular AutoML approaches are compared: (i) "*Random*": Both operations and weights (for the small and fixed MLP) in the designed search space (in Section 4.2.2) are uniformly and randomly set (specifically, random search (Bergstra and Bengio, 2012) is used). (ii) Following (Zoph and Le, 2017), we use reinforcement learning ("*RL*") (Sutton and Barto, 1998) to search the designed space; (iii) "*Bayes*": the designed search space is optimized by HyperOpt (Bergstra et al., 2015), which is a popular Bayesian optimization approach for hyperparameter tuning; and (iv) the proposed Algorithm 2 (denoted "*SIF*") and (v) its variant in which parameter $\boldsymbol{S}$ for IFCs are also optimized with training data (denoted "*SIF(no-auto)*").

Effectiveness Testing RMSEs of the various AutoML approaches are shown in Figure 4.4. Note that MovieLens-10M is not tested as other methods (except *SIF*) are too slow (Figure 4.5). *SIF(no-auto)* is worse than *SIF* as IFCs is purely searched by the training set. Among all methods, the proposed *SIF* is the best. It can find good IFCs, leading to lower testing RMSEs than all other methods under different embedding dimensions.

Search efficiency Efficiency of the various AutoML approaches are compared in Figure 4.5. Top-k testing RMSE is reported, i.e., architectures which achieve top k validation performance

dim	Random	RL	Bayes	SIF
1	concat	concat	min	inner
2	concat	plus	concat	min
4	concat	concat	concat	min
8	plus	plus	plus	inner
16	concat	plus	plus	plus

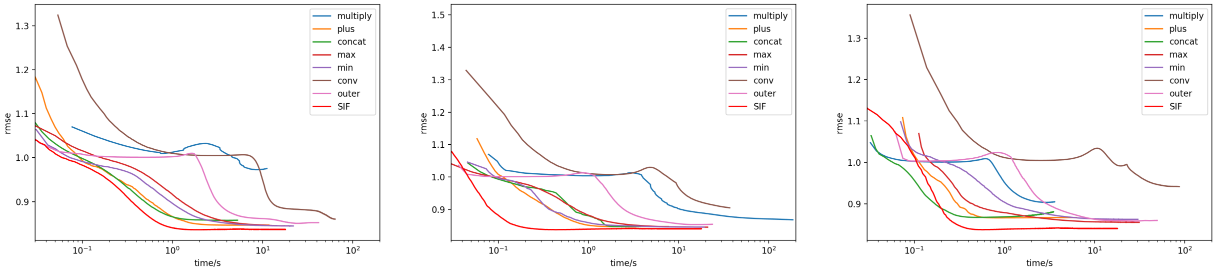


(a) Operations (vector-wise).

(b) Non-linear transformation (element-wise).

(c) Performance of each single operation.

Figure 4.6: (a-b). Searched IFCs on MovienLens-100K with embedded dimension equals 8. (c). Performance comparison between SIF and each single operation on MovieLens-100K.



(a) embedding dimension = 4.

(b) embedding dimension = 8.

(c) embedding dimension = 16.

Figure 4.7: Convergence of various operations on MovieLens-100K data set.

are kept and they are re-trained with RMSE on the testing set is reported. First, due to the large search space introduced by a general approximator, *Gen-approx* can be slower than *Random*, *RL* and *Bayes*. Moreover, since the search space for tensor data is larger than that for matrix data, all algorithms become slower on Youtube. Besides, *SIF* is much faster than all the other methods and has lower testing RMSEs. The gap is larger on the Youtube dataset. Finally, SIF can find IFCs within 5 times of clock times from humans' fine-tuning of existing CFs (Table 4.3).

Table 4.4: The impact of allowing more selected operations in SIF. MovieLens-100K is used.

	embedding dimension = 4		embedding dimension = 8	
k: operations	RMSE	operator	RMSE	operator
1	0.8448	concat	0.8450	max
2	0.8435	concat, max	0.8440	max, plus
3	0.8442	concat, max, multiply	0.8432	max, plus, concat
4	0.8433	concat, max, multiply, plus	0.8437	max, plus, concat, min
5	**0.8432**	concat, max, multiply, plus, min	**0.8431**	max, plus, concat, min, multiply

4.3.4 Case Study: The Searched Interaction Functions (IFCs)

To explain why a lower RMSE can be achieved by the proposed method, we show the searched IFCs by various AutoML methods. Results on MovieLens-100K are shown in Figure 4.6(a). We can see that *Random*, *RL*, *Bayes* and *SIF* generally pick up different operations. Besides, we also show in Figure 4.6(b) the searched nonlinear transformation for each single element. We can see that *SIF* can find more complex transformations than the other methods.

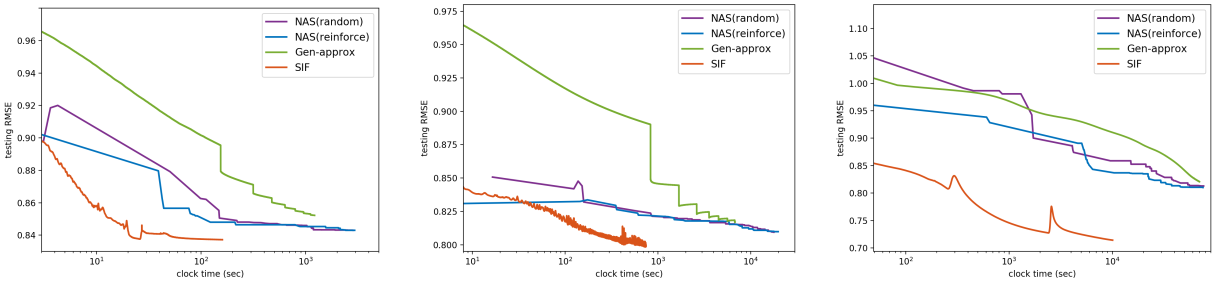
To further demonstrate the need of AutoML and effectiveness of SIF, we show the performance of each single operation in Figure 4.6(c). It can be seen that while some operations can be better than others (e.g., plus is better than conv), there is no clear winner among all operations; and the best operation may depend on the embedded dimensions as well. These verify the need for AutoML. Besides, SIF consistently achieves lower testing RMSEs than all single operations, and converges faster as well (see Figure 4.7). Note that SIF in Figure 4.6(a) may not select the best single operation in Figure 4.6(c), due to the learned nonlinear transformation (Figure 4.6(b)).

4.3.5 Ablation Study

Here, we carry ablation studies on different perspectives of the proposed AutoML method.

Table 4.5: Using a small MLP as element-wise transformation with different activation function (rows) and number of hidden units (columns). MovieLens-100K is used. Testing RMSE is reported.

embedding	activation	number of hidden units				
dimension	function	1	5	10	15	20
4	relu	0.8437	0.8388	**0.8385**	0.8389	0.8396
	sigmoid	0.8440	0.8391	0.8390	0.8395	0.8399
	tanh	0.8439	0.8991	0.8389	0.8393	0.8401
8	relu	0.8385	0.8372	**0.8370**	0.8371	0.8374
	sigmoid	0.8382	0.8375	0.8377	0.8376	0.8378
	tanh	0.8386	0.8376	0.8373	0.8375	0.8377



(a) MovieLens-100K.

(b) MovieLens-1M.

(c) Youtube.

Figure 4.8: Comparison on different search space designs. Embedding dimension is 8.

Different search spaces First, we show the superiority of search space design in SIF by comparing with some common designs in the AutoML literature. The following approaches are compared with SIF:

- Using a MLP as a general approximator ("*Gen-approx*"), as described in Section 4.2.2, to approximate the search space is also compared. The MLP is updated with gradient descent (Bengio, 2000) using the validation set. Since searching network architectures is expensive (Zoph et al., 2017; Zoph and Le, 2017), the structure of the MLP is fixed for *Gen-approx*.
- Standard NAS approach, using MLP to approximate the IFC f. The MLP is optimized with the training data set, while its architecture is searched with the validation set. Two kinds of search algorithms are considered: 1) random search (denoted ' '*NAS(random)*") Bergstra and Bengio (2012) 2) reinforcement learning (denoted "*NAS(reinforce)*") Zoph and Le (2017).

We do not compare with the joint optimization of architecture and weights of the MLP networks, as it is too expensive.

Results are plotted in Figure 4.8. As we can see these general approximation methods are hard to be searched and much slower than SIF. The proposed search space in Section 4.2.2 is not only compact, but also allows efficient one-shot search as discussed in Section 4.2.3.

Allowing more operations In Algorithm 2, we only allow one operation to be selected. Here, we allow more operations by changing $\mathcal{C}_1$ to $\mathcal{C}_k = \{\boldsymbol{\alpha} \,|\, \|\boldsymbol{\alpha}\|_0 = k\}$ where $k \in \{1, \cdots, 5\}$. Results are in Table 4.4. We can see that the testing RMSE can slightly get smaller. However, the model complexity and prediction time can significantly become lager, which should grow linearly with k.

Table 4.6: Using MLP instead of the linear predictor in (4.2). MovieLens-100K is used.

	MLP		linear	
dim	RMSE	operator	RMSE	operator
2	0.8437	concat	**0.8389**	min
4	0.8424	concat	0.8429	min
8	0.8407	plus	0.8468	inner
16	0.8413	multiply	0.8467	plus

Element-wise transformation Recall that in Section 4.2.2, we use a small MLP to approximate an arbitrary element-wise transformation. We inspect the number of hidden units and activation function in such a small MLP here. Results are in Table 4.5. As can be seen, the performance is stable w.r.t different choices of activation functions once the number of hidden units is large enough (i.e., ≥ 5 here). This demonstrates the robustness of our design in the search space (Figure 4.1).

Changing predictor to MLP In the above, we used the linear predictor, i.e., $\boldsymbol{w}$ in (4.2), as the predictor to generate the final estimation. Here, we inspect whether changing the predictor can further boost learning performance. A standard three-layer MLP with 10 hidden units is used here, and results are in Table 4.6. We can see that using a more complex predictor can generally leads to a lower testing RMSE (i.e., dim $= 4, 8, 16$). However, the lowest testing RMSE is still achieved by the linear predictor (i.e., dim $= 2$). This demonstrates that the proposed SIF can achieve the desired performance, and designing a proper predictor is not an easy task.

CHAPTER 5

Optimization Gap in Transformers: Transitioning from Architecture to Optimizer Search

The realms of computer vision and natural language processing have long been dominated by Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These high-performing and efficient architectures are pivotal across various machine learning pipelines, prompting a myriad of studies proposing innovative design modifications.

With the advent of Transformers (Vaswani et al., 2017), their prevalence, particularly in the development of large language models, has seen a notable shift in focus within the research community. Scholars and practitioners are increasingly leaning towards enhancing features related to this architecture, such as optimizers, which are integral components in its refinement. This emphasis on optimizing elements around the existing architectural framework underscores a broader trend of prioritizing advancements in areas like training methodologies, attention mechanisms, and parameter efficiency, among others.

In computer vision, there has recently been a surge of interest in end-to-end Transformers (Akbari et al., 2021; Arnab et al., 2021; Bertasius et al., 2021; Dosovitskiy et al., 2021a; Fan et al., 2021; Liu et al., 2021b; Touvron et al., 2021b) and MLPs (Liu et al., 2021a; Melas-Kyriazi, 2021; Tolstikhin et al., 2021; Touvron et al., 2021a), prompting the efforts to replace hand-wired features or inductive biases with general-purpose neural architectures powered by data-driven training. We envision these efforts may lead to a unified knowledge base that produces versatile representations for different data modalities, simplifying the inference and deployment of deep learning models in various application scenarios.

Despite the appealing potential of moving toward general-purpose neural architectures, the lack of convolution-like inductive biases also challenges the training of vision Transformers (ViTs) and MLPs. When trained on ImageNet (Deng et al., 2009) with the conventional Inception-style data preprocessing (Szegedy et al., 2016), Transformers *"yield modest accuracies of a few percentage points below ResNets of comparable size"* (Dosovitskiy et al., 2021a). To boost the performance, existing works resort to large-scale pre-training (Akbari et al., 2021; Arnab et al., 2021; Dosovitskiy et al., 2021a) and repeated strong data augmentations (Touvron et al., 2021b), resulting in excessive demands of data, computing, and sophisticated tuning of many hyperparameters. For instance, Dosovitskiy et al. (Dosovitskiy et al., 2021a) pre-train ViTs using 304M labeled images, and Touvron et al. (2021b) repeatedly stack four strong image augmentations.

In this chapter, we show ViTs can outperform ResNets (He et al., 2016) of even bigger sizes in both accuracy and various forms of robustness by using a principled optimizer, without the need for large-scale pre-training or strong data augmentations. MLP-Mixers (Tolstikhin et al., 2021) also become on par with ResNets.

We first study the architectures fully trained on ImageNet from the lens of loss landscapes and draw the following findings. First, visualization and Hessian matrices of the loss landscapes reveal that Transformers and MLP-Mixers converge at extremely sharp local minima, whose largest principal curvatures are almost an order of magnitude bigger than ResNets'. Such effect accumulates when the gradients backpropagate from the last layer to the first, and the initial embedding layer suffers the largest eigenvalue of the corresponding sub-diagonal Hessian. Second, the networks all have very small training errors, and MLP-Mixers are more prone to overfitting than ViTs of more parameters (because of the difference in self-attention). Third, ViTs and MLP-Mixers have worse "trainabilities" than ResNets following the neural tangent kernel analyses (Xiao et al., 2020).

Therefore, we need improved learning algorithms to prevent the convergence to a sharp local minimum when it comes to the convolution-free ViTs and MLP-Mixers. The first-order optimizers (e.g., SGD and Adam (Kingma and Ba, 2015)) only seek the model parameters that minimize the training error. They dismiss the higher-order information such as flatness that correlates with generalization (Chaudhari et al., 2017; Jastrzębski et al., 2019; Keskar et al., 2017; Kleinberg et al., 2018; Smith and Le, 2018).

The above study and reasoning lead us to the recently proposed sharpness-aware minimizer (SAM) (Foret et al., 2021b) that explicitly smooths the loss geometry during model training. SAM strives to find a solution whose entire neighborhood has low losses rather than focus on any singleton point. We show that the resultant models exhibit smoother loss landscapes, and their generalization capabilities improve tremendously across different tasks including supervised, adversarial, contrastive, and transfer learning (e.g., +5.3% and +11.0% top-1 accuracy on ImageNet for ViT-B/16 and Mixer-B/16, respectively, with the simple Inception-style preprocessing). The enhanced ViTs achieve better accuracy and robustness than ResNets of similar and bigger sizes when trained from scratch on ImageNet, without large-scale pre-training or strong data augmentations. Moreover, we demonstrate that SAM can even enable ViT to be effectively trained with (momentum) SGD, which usually lies far behind Adam when training Transformers (Zhang et al., 2020).

By analyzing some intrinsic model properties, we observe that SAM increases the sparsity of active neurons (especially for the first few layers), which contribute to the reduced Hessian eigenvalues. The weight norms increase, implying the commonly used weight decay may not be an effective regularization alone. A side observation is that, unlike ResNets and MLP-Mixers, ViTs have extremely sparse active neurons (see Figure 5.2 (right)), revealing the potential for network pruning (Akbari et al., 2021). Another interesting finding is that the improved ViTs appear to have visually more interpretable attention maps. Finally, we draw similarities between SAM and strong augmentations (e.g., mixup) in that they both smooth the average loss geometry and encourage the models to behave linearly between training images.

5.1 Background of Transformer in Vision

We briefly review ViTs, MLP-Mixers, and some related works in this section.

Dosovitskiy et al. (2021a) show that a pure Transformer architecture (Vaswani et al., 2017) can achieve state-of-the-art accuracy on image classification by pre-training it on large datasets such as ImageNet-21k (Deng et al., 2009) and JFT-300M (Sun et al., 2017b). Their vision Transformer (ViT) is a stack of residual blocks, each containing a multi-head self-attention, layer normalization (Ba et al., 2016), and a MLP layer. ViT first embeds an input image $x \in \mathbb{R}^{H \times W \times C}$ into a sequence of features $z \in \mathbb{R}^{N \times D}$ by applying a linear projection over N nonoverlapping image patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where D is the feature dimension, P is the patch resolution, and $N = HW/P^2$ is the sequence length. The self-attention layers in ViT are global and do not possess the locality and translation equivariance of convolutions. ViT is compatible with the popular architectures in NLP (Devlin et al., 2018; Radford et al., 2018) and, similar to its NLP counterparts, requires pre-training over massive datasets (Akbari et al., 2021; Arnab et al., 2021; Dosovitskiy et al., 2021a) or strong data augmentations (Touvron et al., 2021b). Some works specialize the ViT architectures for visual data (Bertasius et al., 2021; Fan et al., 2021; Liu et al., 2021b; Yuan et al., 2021).

More recent works find that the self-attention in ViT is not vital for performance, resulting in several architectures exclusively based on MLPs (Liu et al., 2021a; Melas-Kyriazi, 2021; Tolstikhin et al., 2021; Touvron et al., 2021a). Here we take MLP-Mixer (Tolstikhin et al., 2021) as an example. MLP-Mixer shares the same input layer as ViT; namely, it partitions an image into a sequence of nonoverlapping patches/tokens. It then alternates between token and channel MLPs, where the former allows feature fusion from different spatial locations.

We focus on ViTs and MLP-Mixers. We denote by “S” and “B” the small and base model sizes, respectively, and by an integer the image patch resolution. For instance, ViT-B/16 is the base ViT model taking as input a sequence of 16×16 patches.

Table 5.1: Number of parameters, NTK condition number κ, Hessian dominate eigenvalue λ_{max}, training error at convergence L_{train}, average flatness $L^{\mathcal{N}}_{train}$, accuracy on ImageNet, and accuracy/robustness on ImageNet-C. ViT and MLP-Mixer suffer divergent κ and converge at sharp regions; SAM rescues that and leads to better generalization.

	ResNet-152	ResNet-152-SAM	ViT-B/16	ViT-B/16-SAM	Mixer-B/16	Mixer-B/16-SAM
#Params	60M		87M		59M	
NTK κ [†]	2801.6		4205.3		14468.0	
Hessian λ_{max}	179.8	**42.0**	738.8	**20.9**	1644.4	**22.5**
L_{train}	**0.86**	0.90	**0.65**	0.82	**0.45**	0.97
$L^{\mathcal{N}}_{train}$ [⋆]	2.39	**2.16**	6.66	**0.96**	7.78	**1.01**
ImageNet (%)	78.5	**79.3**	74.6	**79.9**	66.4	**77.4**
ImageNet-C (%)	50.0	**52.2**	46.6	**56.5**	33.8	**48.8**

[†] As it is prohibitive to compute the exact NTK, we approximate the value by averaging over its sub-diagonal blocks. We average the results for 1,000 random noises when calculating $L^{\mathcal{N}}_{train}$.

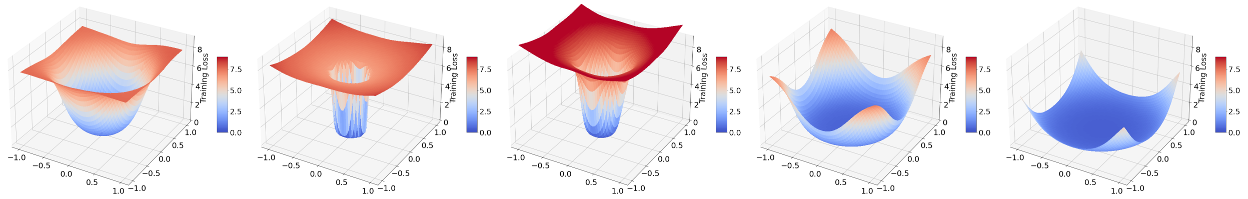


Figure 5.1: Cross-entropy loss landscapes of ResNet-152, ViT-B/16, Mixer-B/16, ViT-B/16-SAM, and Mixer-B/16-SAM (from left to right). ViT and MLP-Mixer converge to sharper regions than ResNet when trained on ImageNet with the basic Inception-style preprocessing. SAM, a sharpness-aware optimizer, significantly smooths the landscapes.

5.2 ViTs and MLP-Mixers Converge at Sharp Local Minima

The current training recipe of ViTs, MLP-Mixers, and related convolution-free architectures relies heavily on massive pre-training (Akbari et al., 2021; Arnab et al., 2021; Dosovitskiy et al., 2021a) or a bag of strong data augmentations (Cubuk et al., 2019, 2020; Tolstikhin et al., 2021; Touvron et al., 2021b; Yun et al., 2019; Zhang et al., 2018a). It highly demands data and computing, and leads to many hyperparameters to tune. Existing works report that ViTs yield inferior accuracy to the ConvNets of similar size and throughput when trained from scratch on ImageNet without the combination of those advanced data augmentations, despite using various regularization techniques (e.g., large weight decay, Dropout (Srivastava et al., 2014), etc.). For instance, ViT-B/16 (Dosovitskiy et al., 2021a) gives rise to 74.6% top-1 accuracy on the ImageNet validation set (224 image resolution), compared with 78.5% of ResNet-152 (He et al., 2016). Mixer-B/16 (Tolstikhin et al., 2021) performs even worse (66.4%). There also exists a large gap between ViTs and ResNets in robustness tests (see Table 5.2 for details).

Moreover, Chen et al. (2021d) find that the gradients can spike and cause a sudden accuracy dip when training ViTs, and Touvron et al. (2021b) report the training is sensitive to initialization and hyperparameters. These all point to optimization problems. In this chapter, we investigate the loss landscapes of ViTs and MLP-Mixers to understand them from the optimization perspective, intending to reduce their dependency on the large-scale pre-training or strong data augmentations.

ViTs and MLP-Mixers converge at extremely sharp local minima It has been extensively studied that the convergence to a flat region whose curvature is small benefits the generalization of neural networks (Chaudhari et al., 2017; Chen and Hsieh, 2020; Jastrzębski et al., 2019; Keskar et al., 2017; Kleinberg et al., 2018; Smith and Le, 2018; Zela et al., 2020a). Following Li et al. (2018), we plot the loss landscapes at convergence when ResNets, ViTs, and MLP-Mixers are trained from scratch on ImageNet with the basic Inception-style preprocessing (Szegedy et al., 2016). As shown in Figure 5.1, ViTs and MLP-Mixers converge at much sharper regions than ResNets. Besides, we calculate the training error under Gaussian perturbations on the model parameters $L_{train}^{\mathcal{N}} = \mathbb{E}_{\epsilon\sim\mathcal{N}}[L_{train}(w + \epsilon)]$ in Table 5.1, which reveals the *average* flatness. Although ViT-B/16 and Mixer-B/16 achieve lower training error L_{train} than that of ResNet-152, their loss values after random weight perturbation become much higher. We further validate the results by computing the dominate Hessian eigenvalue λ_{max}, which is a mathematical evaluation of the *worst-case* landscape curvature. The λ_{max} values of ViT and MLP-Mixer are orders of magnitude larger than that of ResNet, and MLP-Mixer suffers the largest curvature among the three species (see Section 5.3.4 for a detailed analysis).

Small training errors This convergence at sharp regions coincides with the training dynamics shown in Figure 5.2 (left). Although Mixer-B/16 has fewer parameters than ViT-B/16 (59M vs. 87M), it has a smaller training error (also see L_{train} in Table 5.1) but much worse test accuracy, implying that using the cross-token MLP to learn the interplay across image patches is more prone to overfitting than ViTs' self-attention mechanism whose behavior is restricted by a softmax. To validate this statement, we simply remove the softmax in ViT-B/16, such that the query and key matrices can freely interact with each other. Although having lower L_{train} (0.56 vs. 0.65), the obtained ViT-B/16-Free performs much worse than the original ViT-B/16 (70.5% vs. 74.6%). Its $L_{train}^{\mathcal{N}}$ and λ_{max} are 7.01 and 1236.2, revealing that ViT-B/16-Free converges to a sharper region than ViT-B/16 ($L_{train}^{\mathcal{N}}$ is 6.66 and λ_{max} is 738.8) both on average and in the worst-case direction. Such a difference probably explains why it is easier for MLP-Mixers to get stuck in sharp local minima.

ViTs and MLP-Mixers have worse trainability Furthermore, we discover that ViTs and MLP-Mixers suffer poor trainabilities, defined as the effectiveness of a network to be optimized by gradient descent (Burkholz and Dubatovka, 2019; Shin and Karniadakis, 2020; Xiao et al., 2020). Xiao et al. (2020) show that the trainability of a neural network can

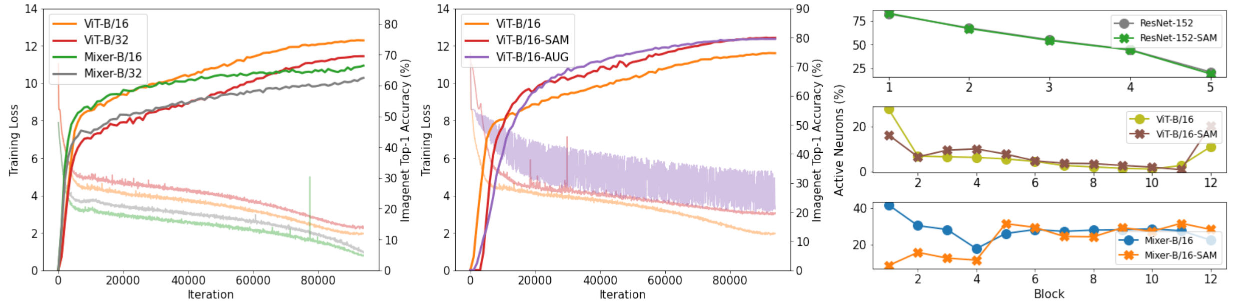


Figure 5.2: **Left** and **Middle**: ImageNet training error and validation accuracy vs. iteration for ViTs and MLP-Mixers. **Right**: Percentage of active neurons for ResNet-152, ViT-B/16, and Mixer-B/16.

be characterized by the condition number of the associated neural tangent kernel (NTK), $\Theta(x, x') = J(x)J(x')^T$, where J is the Jacobian matrix. Denoting by $\lambda_1 \geq \cdots \geq \lambda_m$ the eigenvalues of NTK Θ_{train}, the smallest eigenvalue λ_m converges exponentially at a rate given by the condition number $\kappa = \lambda_1/\lambda_m$. If κ diverges then the network will become untrainable (Chen et al., 2021b; Xiao et al., 2020). As shown in Table 5.1, κ is pretty stable for ResNets, echoing previous results that ResNets enjoy superior trainability regardless of the depth (Li et al., 2018; Yang and Schoenholz, 2017). However, we observe that the condition number diverges when it comes to ViT and MLP-Mixer, confirming that the training of ViTs desires extra care (Chen et al., 2021d; Touvron et al., 2021b).

5.3 A Principled Optimizer for Convolution-Free Architectures

The commonly used first-order optimizers (e.g., SGD (Nesterov, 1983), Adam (Kingma and Ba, 2015)) only seek to minimize the training loss $L_{train}(w)$. They usually dismiss the higher-order information such as curvature that correlates with the generalization (Chaudhari et al., 2017; Dziugaite and Roy, 2017; Keskar et al., 2017). However, the objective L_{train} for deep neural networks are highly non-convex, making it easy to reach near-zero training error but high generalization error L_{test} during evaluation, let alone their robustness when the test sets have different distributions (Hendrycks and Dietterich, 2019; Hendrycks et al., 2020). ViTs and MLPs amplify such drawbacks of first-order optimizers due to the lack of inductive bias for visual data, resulting in excessively sharp loss landscapes and poor generalization, as shown in the previous section. We hypothesize that smoothing the loss landscapes at convergence can significantly improve the generalization ability of those convolution-free architectures, leading us to the recently proposed sharpness-aware minimizer (SAM) (Foret et al., 2021b) that explicitly avoids sharp minima.

5.3.1 SAM: Overview

Intuitively, SAM (Foret et al., 2021b) seeks to find the parameter w whose entire neighbours have low training loss L_{train} by formulating a minimax objective:

$$\min_{w} \max_{\|\epsilon\|_2 \leq \rho} L_{train}(w+\epsilon), \tag{5.1}$$

where ρ is the size of the neighbourhood ball. Without loss of generality, here we use l_2 norm for its strong empirical results (Foret et al., 2021b) and omit the regularization term for simplicity. Since the exact solution of the inner maximization $\epsilon^\star = \arg\max_{\|\epsilon\|_2 \leq \rho} L_{train}(w+\epsilon)$ is hard to obtain, they employ an efficient first-order approximation:

$$\hat{\epsilon}(w) = \arg\max_{\|\epsilon\|_2 \leq \rho} L_{train}(w) + \epsilon^T \nabla_w L_{train}(w) = \rho \nabla_w L_{train}(w) / \|\nabla_w L_{train}(w)\|_2. \tag{5.2}$$

Under the l_2 norm, $\hat{\epsilon}(w)$ is simply a scaled gradient of the current weight w. After computing $\hat{\epsilon}$, SAM updates w based on the sharpness-aware gradient $\nabla_w L_{train}(w)|_{w+\hat{\epsilon}(w)}$.

5.3.2 Sharpness-aware optimization improves ViTs and MLP-Mixers

We train ViTs and MLP-Mixers with no large-scale pre-training or strong data augmentations. We directly apply SAM to the original ImageNet training pipeline of ViTs (Dosovitskiy et al., 2021a) without changing any hyperparameters. The pipeline employs the basic Inception-style preprocessing (Szegedy et al., 2016). The original training setup of MLP-Mixers (Tolstikhin et al., 2021) includes a combination of strong data augmentations, and we replace it with the same Inception-style preprocessing for a fair comparison. Note that we perform grid search for the learning rate, weight decay, Dropout *before* applying SAM.

Smoother regions around the local minima Thanks to SAM, both ViTs and MLP-Mixers converge at much smoother regions, as shown in Figure 5.1. Moreover, both the average and the worst-case curvature, i.e., $L^{\mathcal{N}}_{train}$ and λ_{max}, decrease dramatically (see Table 5.1).

Higher accuracy What comes along is tremendously improved generalization performance. On ImageNet, SAM boosts the top-1 accuracy of ViT-B/16 from 74.6% to 79.9%, and Mixer-B/16 from 66.4% to 77.4%. For comparison, the improvement on a similarly sized ResNet-152 is 0.8%. Empirically, *the degree of improvement negatively correlates with the constraints of inductive biases built into the architecture.* ResNets with inherent translation equivalence and locality benefit less from landscape smoothing than the attention-based ViTs. MLP-Mixers gain the most from the smoothed loss geometry. In Table 5.3, we further train two hybrid models (Dosovitskiy et al., 2021a) to validate this observation, where the Transformer takes the feature map extracted from a ResNet-50 as the input sequence. The improvement brought by SAM decreases after we introduce the convolution to ViT, for instance, +2.7% for R50-B/16 compared to +5.3% for ViT-B/16. Moreover, SAM brings larger improvements to the models of larger capacity (e.g., +4.1% for Mixer-S/16 vs. +11.0% for Mixer-B/16) and longer patch sequence (e.g., +2.1% for ViT-S/32 vs. +5.3% for ViT-S/8). Please see Table 5.2 for more results.

SAM can be easily applied to common base optimizers. Besides Adam, we also apply SAM on top of the (momentum) SGD that usually performs much worse than Adam when training Transformers (Zhang et al., 2020). As expected, we find that under the same training budget (300 epochs), the ViT-B/16 trained with SGD only achieves 71.5% accuracy on ImageNet, whereas Adam achieves 74.6%. Surprisingly, SGD + SAM can push the result to 79.1%, which is a huge +7.6% absolute improvement. Although Adam + SAM is still higher (79.9%), their gap largely shrinks.

Better robustness We also evaluate the models' robustness using ImageNet-R (Hendrycks et al., 2020) and ImageNet-C (Hendrycks and Dietterich, 2019) and find even bigger impacts of the smoothed loss landscapes. On ImageNet-C, which corrupts images by noise, bad weather, blur, etc., we report the average accuracy against 19 corruptions across five levels. As shown in Tables 5.1 and 5.2, the accuracies of ViT-B/16 and Mixer-B/16 increase by 9.9% and 15.0% (which are 21.2% and 44.4% *relative* improvements), after SAM smooths their converged local regions. In comparison, SAM improves the accuracy of ResNet-152 by 2.2% (4.4% *relative* improvement). We can see that SAM enhances the robustness even more than the *relative* clean accuracy improvements (7.1%, 16.6%, and 1.0% for ViT-B/16, Mixer-B/16, and ResNet-152, respectively).

5.3.3 ViTs outperform ResNets without pre-training or strong augmentations

The performance of an architecture is often conflated with the training strategies (Bello et al., 2021), where data augmentations play a key role (Chen et al., 2021c; Cubuk et al., 2019, 2020; Xie et al., 2020; Zhang et al., 2018a). However, the design of augmentations requires substantial domain expertise and may not translate between images and videos, for instance. Thanks to the principled sharpness-aware optimizer, we can remove the advanced augmentations and focus on the architectures themselves.

When trained from scratch on ImageNet with SAM, *ViTs outperform ResNets of similar and greater sizes (also comparable throughput at inference)* regarding both clean accuracy (on ImageNet (Deng et al., 2009), ImageNet-ReaL (Beyer et al., 2020a), and ImageNet V2 (Recht et al., 2019a)) and robustness (on ImageNet-R (Hendrycks et al., 2020) and ImageNet-C (Hendrycks and Dietterich, 2019)). ViT-B/16 achieves 79.9%, 26.4%, and 56.6% top-1 accuracy on ImageNet, ImageNet-R, and ImageNet-C, while the counterpart numbers

Table 5.2: Performance of ResNets, ViTs, and MLP-Mixers trained from scratch on ImageNet with SAM (improvement over the vanilla model is shown in the parentheses). We use the Inception-style preprocessing (with resolution 224) rather than a combination of strong data augmentations.

Model	**#params**	**Throughput (img/sec/core)**	**ImageNet**	**ReaL**	**V2**	**ImageNet-R**	**ImageNet-C**
he2016resnet							
ResNet-50-SAM	25M	2161	76.7 (+0.7)	83.1 (+0.7)	64.6 (+1.0)	23.3 (+1.1)	46.5 (+1.9)
ResNet-101-SAM	44M	1334	78.6 (+0.8)	84.8 (+0.9)	66.7 (+1.4)	25.9 (+1.5)	51.3 (+2.8)
ResNet-152-SAM	60M	935	79.3 (+0.8)	84.9 (+0.7)	67.3 (+1.0)	25.7 (+0.4)	52.2 (+2.2)
ResNet-50x2-SAM	98M	891	79.6 (+1.5)	85.3 (+1.6)	67.5 (+1.7)	26.0 (+2.9)	50.7 (+3.9)
ResNet-101x2-SAM	173M	519	80.9 (+2.4)	86.4 (+2.4)	69.1 (+2.8)	27.8 (+3.2)	54.0 (+4.7)
ResNet-152x2-SAM	236M	356	81.1 (+1.8)	86.4 (+1.9)	69.6 (+2.3)	28.1 (+2.8)	55.0 (+4.2)
Vision Transformer							
ViT-S/32-SAM	23M	6888	70.5 (+2.1)	77.5 (+2.3)	56.9 (+2.6)	21.4 (+2.4)	46.2 (+2.9)
ViT-S/16-SAM	22M	2043	78.1 (+3.7)	84.1 (+3.7)	65.6 (+3.9)	24.7 (+4.7)	53.0 (+6.5)
ViT-S/14-SAM	22M	1234	78.8 (+4.0)	84.8 (+4.5)	67.2 (+5.2)	24.4 (+4.7)	54.2 (+7.0)
ViT-S/8-SAM	22M	333	81.3 (+5.3)	86.7 (+5.5)	70.4 (+6.2)	25.3 (+6.1)	55.6 (+8.5)
ViT-B/32-SAM	88M	2805	73.6 (+4.1)	80.3 (+5.1)	60.0 (+4.7)	24.0 (+4.1)	50.7 (+6.7)
ViT-B/16-SAM	87M	863	79.9 (+5.3)	85.2 (+5.4)	67.5 (+6.2)	26.4 (+6.3)	56.5 (+9.9)
MLP-Mixer							
Mixer-S/32-SAM	19M	11401	66.7 (+2.8)	73.8 (+3.5)	52.4 (+2.9)	18.6 (+2.7)	39.3 (+4.1)
Mixer-S/16-SAM	18M	4005	72.9 (+4.1)	79.8 (+4.7)	58.9 (+4.1)	20.1 (+4.2)	42.0 (+6.4)
Mixer-S/8-SAM	20M	1498	75.9 (+5.7)	82.5 (+6.3)	62.3 (+6.2)	20.5 (+5.1)	42.4 (+7.8)
Mixer-B/32-SAM	60M	4209	72.4 (+9.9)	79.0 (+10.9)	58.0 (+10.4)	22.8 (+8.2)	46.2 (12.4)
Mixer-B/16-SAM	59M	1390	77.4 (+11.0)	83.5 (+11.4)	63.9 (+13.1)	24.7 (+10.2)	48.8 (+15.0)
Mixer-B/8-SAM	64M	466	79.0 (+10.4)	84.4 (+10.1)	65.5 (+11.6)	23.5 (+9.2)	48.9 (+16.9)

for ResNet-152 are 79.3%, 25.7%, and 52.2%, respectively (see Table 5.2). The gaps between ViTs and ResNets are even wider for small architectures. ViT-S/16 outperforms a similarly sized ResNet-50 by 1.4% on ImageNet and 6.5% on ImageNet-C. MLP-Mixers' performance is also significantly improved.

Table 5.3: Accuracy and robustness of two hybrid architectures.

Model	**#params**	**ImageNet (%)**	**ImageNet-C (%)**
R50-S/16	34M	79.8	53.4
R50-S/16-SAM		81.0 (+1.2)	57.2 (+3.8)
R50-B/16	99M	79.7	54.4
R50-B/16-SAM		82.4 (+2.7)	61.0 (+6.6)

5.3.4 Intrinsic changes after SAM

We take a deeper look into the models to understand how they intrinsically change to reduce the Hessian' eigenvalue λ_{max} and what the changes imply in addition to the enhanced generalization.

Smoother loss landscapes for every network component In Table 5.4, we break down the Hessian of the whole architecture into small diagonal blocks of Hessians concerning each set of parameters, attempting to analyze what specific components cause the blowing up of λ_{max} in the models trained without SAM. We observe that shallower layers have larger Hessian eigenvalues λ_{max}, and the first linear embedding layer incurs the sharpest geometry. This agrees with the finding in (Chen et al., 2021d) that spiking gradients happen early in the embedding layer. Additionally, the multi-head self-attention (MSA) in ViTs and the Token MLPs in MLP-Mixers, both of which mix information across spatial locations, have comparably lower λ_{max} than the other network components. SAM consistently reduces the λ_{max} of all network blocks.

We can gain insights into the above findings by the recursive formulation of Hessian matrices for MLPs (Botev et al., 2017). Let h_k and a_k be the pre-activation and post-activation values for layer k, respectively. They satisfy $h_k = W_k a_{k-1}$ and $a_k = f_k(h_k)$, where

Table 5.4: Dominant eigenvalue λ_{max} of the sub-diagonal Hessians for different network components, and norm of the model parameter w and the post-activation a_k of block k. Each ViT block consists of a MSA and a MLP, and MLP-Mixer alternates between a token MLP a channel MLP. Shallower layers have larger λ_{max}. SAM smooths every component.

Model	λ_{max} **of diagonal blocks of Hessian**							$\|w\|_2$	$\|a_1\|_2$	$\|a_6\|_2$	$\|a_{12}\|_2$
	Embedding	MSA/ Token MLP	MLP/ Channel MLP	Block1	Block6	Block12	Whole				
ViT-B/16	300.4	179.8	281.4	44.4	32.4	26.9	738.8	269.3	104.9	104.3	138.1
ViT-B/16-SAM	3.8	8.5	9.6	1.7	1.7	1.5	20.9	353.8	117.0	120.3	97.2
Mixer-B/16	1042.3	95.8	417.9	239.3	41.2	5.1	1644.4	197.6	96.7	135.1	74.9
Mixer-B/16-SAM	18.2	1.4	9.5	4.0	1.1	0.3	22.5	389.9	110.9	176.0	216.1

W_k is the weight matrix and f_k is the activation function (GELU (Hendrycks and Gimpel, 2020) in MLP-Mixers). Here we omit the bias term for simplicity. The diagonal block of Hessian matrix H_k with respect to W_k can be recursively calculated as:

$$H_k = (a_{k-1}a_{k-1}^T) \otimes \mathcal{H}_k, \quad \mathcal{H}_k = B_k W_{k+1}^T \mathcal{H}_{k+1} W_{k+1} B_k + D_k, \tag{5.3}$$

$$B_k = \mathrm{diag}(f_k'(h_k)), \qquad D_k = \mathrm{diag}(f_k''(h_k)\frac{\partial L}{\partial a_k}), \tag{5.4}$$

where $\otimes$ is the Kronecker product, $\mathcal{H}_k$ is the pre-activation Hessian for layer k, and L is the objective function. Therefore, the Hessian norm accumulates as the recursive formulation backpropagates to shallow layers, explaining why the first block has much larger λ_{max} than the last block in Table 5.4.

Greater weight norms After applying SAM, we find that in most cases, the norm of the post-activation value a_{k-1} and the weight W_{k+1} become even bigger (see Table 5.4), indicating that the commonly used weight decay may not effectively regularize ViTs and MLP-Mixers (see Section 5.4.5 for further verification when we vary the weight decay strength).

Table 5.5: Data augmentations, SAM, and their combination applied to different model architectures trained on ImageNet and its subsets from scratch.

Dataset	ResNet-152				ViT-B/16				Mixer-B/16			
	Vanilla	SAM	AUG	SAM + AUG	Vanilla	SAM	AUG	SAM + AUG	Vanilla	SAM	AUG	SAM + AUG
ImageNet	78.5	79.3	78.8	78.9	74.6	79.9	79.6	81.5	66.4	77.4	76.5	78.1
i1k (1/2)	74.2	75.6	75.1	75.5	64.9	75.4	73.1	75.8	53.9	71.0	70.4	73.1
i1k (1/4)	68.0	70.3	70.2	70.6	52.4	66.8	63.2	65.6	37.2	62.8	61.0	65.8
i1k (1/10)	54.6	57.1	59.2	59.5	32.8	46.1	38.5	45.7	21.0	43.5	43.0	51.0

Sparser active neurons in MLP-Mixers Given the recursive formulation Equation (5.3), we identify another intrinsic measure of MLP-Mixers that contribute to the Hessian: the number of activated neurons. Indeed, B_k is determined by the activated neurons whose values are greater than zero, since the first-order derivative of GELU becomes much smaller when the input is negative. As a result, the number of active GELU neurons is directly connected to the Hessian norm. Figure 5.2 (right) shows the proportion of activated neurons for each block, counted using 10% of the ImageNet training set. We can see that SAM greatly reduces the proportion of activated neurons for the first few layers of the Mixer-B/16, pushing them to much sparser states. This result also suggests the potential redundancy of image patches.

ViTs' active neurons are highly sparse Although Equations (5.3) and (5.4) only involve MLPs, we still observe a decrease of activated neurons in the first layer of ViTs (but not as significant as in MLP-Mixers). More interestingly, we find that the proportion of active neurons in ViT is much smaller than another two architectures — given an input image, less than 10% neurons have values greater than zero for most layers (see Figure 5.2 (right)). In other words, ViTs offer a huge potential for network pruning. This sparsity may also explain why one Transformer can handle multi-modality signals (vision, text, and audio) (Akbari et al., 2021).

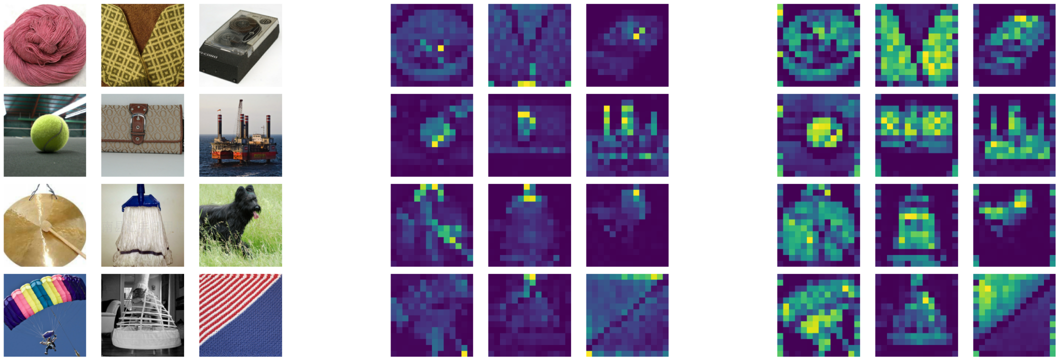


Figure 5.3: Raw images (**Left**) and attention maps of ViT-S/16 with (**Right**) and without (**Middle**) sharpness-aware optimization.

Visually improved attention maps in ViTs We visualize ViT-S/16's attention map of the classification token averaged over the last multi-head attentions in Figure 5.3 following Caron et al. (2021). Interestingly, the ViT model optimized with SAM appears to possess visually improved attention map compared with the one trained via the vanilla AdamW optimizer.

5.3.5 SAM vs. strong augmentations

Previous sections show that SAM can improve the generalization (and robustness) of ViTs and MLP-Mixers. Meanwhile, another paradigm to train these models on ImageNet from scratch is to stack multiple strong augmentations (Tolstikhin et al., 2021; Touvron et al., 2021a,b). Hence, it is interesting to study the differences and similarities between the models trained by SAM and by using strong data augmentations. For the augmentation experiments, we follow Tolstikhin et al. (2021)'s pipeline that includes mixup (Zhang et al., 2018a) and RandAugment (Cubuk et al., 2020).

Generalization. Table 5.5 shows the results of strong data augmentation, SAM, and their combination on ImageNet. Each row corresponds to a training set of a different fraction of ImageNet-1k. SAM benefits ViT-B/16 and Mixer-B/16 more than the strong data

Table 5.6: Comparison between ViT-B/16-SAM and ViT-B/16-AUG. R denotes the missing rate under linear interpolation.

Model	λ_{max}	L_{train}	$L^{\mathcal{N}}_{train}$	$R(\downarrow)$
ViT-B/16	738.8	0.65	6.66	57.9%
ViT-B/16-SAM	20.9	0.82	0.96	39.6%
ViT-B/16-AUG	1659.3	0.85	1.23	21.4%

augmentations, especially when the training set is small. For instance, when the training set contains only 1/10 of ImageNet training images, ViT-B/16-SAM outperforms ViT-B/16-AUG by 7.6%. Apart from the improved validation accuracy, we also observe that both SAM and strong augmentations increase the training error (see Figure 5.2 (Middle) and Table 5.6), indicating their regularization effects. However, they have distinct training dynamics as the loss curve for ViT-B/16-AUG is much nosier than ViT-B/16-SAM.

Sharpness at convergence Another intriguing question is as follows. Can augmentations also smooth the loss geometry similarly to SAM? To answer it, we also plot the landscape of ViT-B/16-AUG in Figure 5.4 and compute its Hessian λ_{max} together with the average flatness $L^{\mathcal{N}}_{train}$ in Table 5.6. Surprisingly, strong augmentations even enlarge the λ_{max}. However, like SAM, augmentations make ViT-B/16-AUG smoother and achieve a significantly smaller training error under random Gaussian perturbations than ViT-B/16. These results show that both SAM and augmentations make the loss landscape flat *on average*. The difference is that SAM enforces the smoothness by reducing the largest curvature via a minimax formulation to optimize the *worst-case* scenario, while augmentations ignore the worse-case curvature and instead smooth the landscape over the directions induced by the augmentations.

Interestingly, besides the similarity in smoothing the loss curvature on average, we also discover that SAM-trained models possess "linearality" resembling the property manually injected by the mixup augmentation. Following Zhang et al. (2018a), we compute the

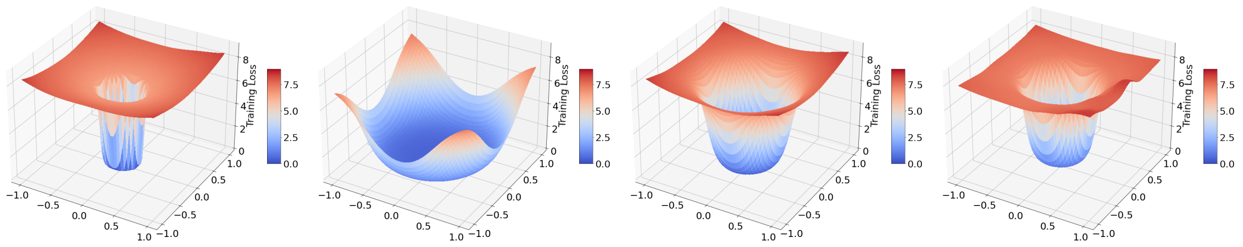


Figure 5.4: Cross-entropy loss landscapes of ViT-B/16, ViT-B/16-SAM, ViT-B/16-AUG, and ViT-B/16-21k (from left to right). Strong augmentations and large-scale pre-training can also smooth the curvature.

prediction error in-between training data in Table 5.6, where a prediction y is counted as a miss if it does not belong to $\{y_i, y_j\}$ evaluated at $x = 0.5x_i + 0.5x_j$. We observe that SAM greatly reduces the missing rate (R) compared with the vanilla baseline, showing a similar effect to mixup that explicitly encourages such linearity.

5.4 Ablation Studies

In this section, we provide a more comprehensive study about SAM's effect on various vision models and under different training setups.

5.4.1 When scaling the training set size

Previous studies scale up training data to show massive pre-training trumps inductive biases (Dosovitskiy et al., 2021a; Tolstikhin et al., 2021). Here we show SAM further enables ViTs and MLP-Mixers to handle small-scale training data well. We randomly sample 1/4 and 1/2 images from each ImageNet class to compose two smaller-scale training sets, i.e., i1k (1/4) and i1k (1/2) with 320,291 and 640,583 images, respectively. We also use ImageNet-21k to pre-train the models with SAM, followed by fine-tuning on ImageNet-1k without SAM. The ImageNet validation set remains intact. SAM can still bring improvement when pre-trained

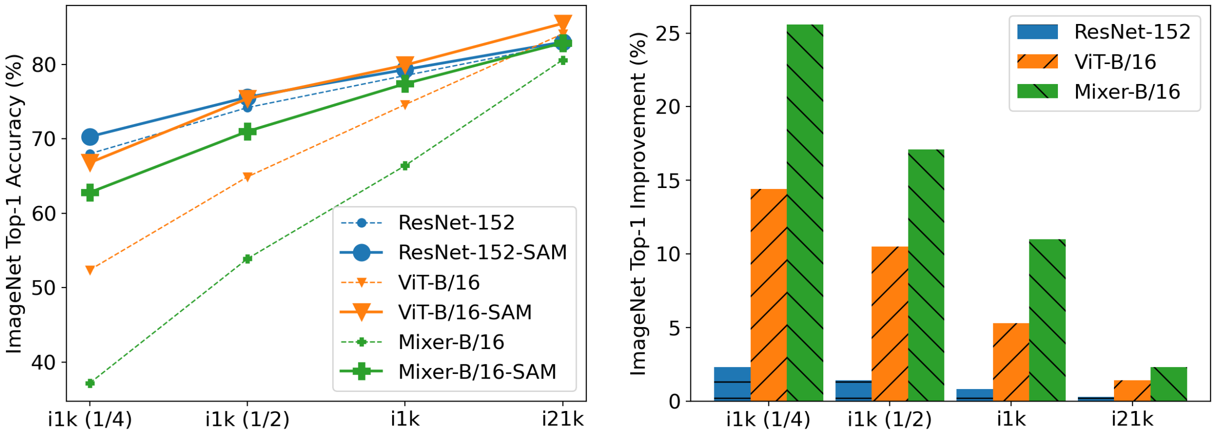


Figure 5.5: ImageNet accuracy (**Left**) and improvement (**Right**) brought by SAM.

on ImageNet-21k (+0.3%, +1.4%, and 2.3% for ResNet-152, ViT-B/16, and Mixer-B/16, respectively).

As expected, fewer training examples amplify the drawback of ViTs and MLP-Mixers' lack of the convolutional inductive bias — their accuracies decline much faster than ResNets' (see Figure 5.5 and the corresponding numbers in Table 5.5).

However, SAM can drastically rescue ViTs and MLP-Mixers' performance decrease on smaller training sets. Figure 5.5 (right) shows that *the improvement brought by SAM over vanilla SGD training is proportional to the number of training images.* When trained on i1k (1/4), it boosts ViT-B/16 and Mixer-B/16 by 14.4% and 25.6%, escalating their results to 66.8% and 62.8%, respectively. It also tells that ViT-B/16-SAM matches the performance of ResNet-152-SAM even with only 1/2 ImageNet training data.

5.4.2 When SAM Meets Adversarial Training

Interestingly, SAM and adversarial training are both minimax problems except that SAM's inner maximization is with respect to the network weights, while the latter concerns about the input for defending contrived attack (Madry et al., 2018b; Wong et al., 2020). Moreover, similar to SAM, Shafahi et al. (2019) suggest that adversarial training can flatten and smooth the loss landscape. In light of these connections, we study ViTs and MLP-Mixers under the adversarial training framework (Madry et al., 2018b; Wu et al., 2020). We use the fast adversarial training (Wong et al., 2020) (FGSM with random start) with the l_∞ norm and maximum per-pixel change 2/255 during training. All the hyperparameters remain the same as the vanilla supervised training. When evaluating the adversarial robustness, we use the PGD attack (Madry et al., 2018b) with the same maximum per-pixel change 2/255. The total number of attack steps is 10, and the step size is 0.25/255. To incorporate SAM, we formulate a three-level objective:

$$\min_{w} \max_{\epsilon \in \mathbb{S}_{sam}} \max_{\delta \in \mathbb{S}_{adv}} L_{train}(w + \epsilon, x + \delta, y), \tag{5.5}$$

where $\mathbb{S}_{sam}$ and $\mathbb{S}_{adv}$ denote the allowed perturbation norm balls for the model parameter w and input image x, respectively. Note that we can simultaneously obtain the gradients for computing ϵ and δ by backpropagation only once. To lower the training cost, we use fast adversarial training (Wong et al., 2020) with the l_∞ norm for δ, and the maximum per-pixel change is set as 2/255.

Table 5.7 (see Appendices) evaluates the models' clean accuracy, real-world robustness, and adversarial robustness (under 10-step PGD attack (Madry et al., 2018b)). It is clear that the landscape smoothing significantly improves the convolution-free architectures for both clean and adversarial accuracy. However, we observe a slight accuracy decrease on clean images for ResNets despite gain for robustness. Similar to our previous observations, *ViTs surpass similar-size ResNets when adversarially trained on ImageNet with Inception-style preprocessing for both clean accuracy and adversarial robustness.*

5.4.3 When SAM Meets Contrastive Learning

In addition to data augmentations and large-scale pre-training, another notable way of improving a neural model's generalization is (supervised) contrastive learning (Caron et al.,

Table 5.7: Comparison under the adversarial training framework on ImageNet (numbers in the parentheses denote the improvement over the standard adversarial training without SAM). With similar model size and throughput, ViTs-SAM can still outperform ResNets-SAM for clean accuracy and adversarial robustness.

Model	#params	Throughput (img/sec/core)	ImageNet	Real	V2	PGD-10	ImageNet-R	ImageNet-C
he2016resnet								
ResNet-50-SAM	25M	2161	70.1 (-0.7)	77.9 (-0.3)	56.6 (-0.8)	54.1 (+0.9)	27.0 (+0.9)	42.7 (-0.1)
ResNet-101-SAM	44M	1334	73.6 (-0.4)	81.0 (+0.1)	60.4 (-0.6)	58.8 (+1.4)	29.5 (+0.6)	46.9 (+0.3)
ResNet-152-SAM	60M	935	75.1 (-0.4)	82.3 (+0.2)	62.2 (-0.4)	61.0 (+1.8)	30.8 (+1.4)	49.1 (+0.6)
Vision Transformer								
ViT-S/16-SAM	22M	2043	73.2 (+1.2)	80.7 (+1.7)	60.2 (+1.4)	58.0 (+5.2)	28.4 (+2.4)	47.5 (+1.6)
ViT-B/32-SAM	88M	2805	69.9 (+3.0)	76.9 (+3.4)	55.7 (+2.5)	54.0 (+6.4)	26.0 (+3.0)	46.4 (+3.0)
ViT-B/16-SAM	87M	863	76.7 (+3.9)	82.9 (+4.1)	63.6 (+4.3)	62.0 (+7.7)	30.0 (+4.9)	51.4 (+5.0)
MLP-Mixer								
Mixer-S/16-SAM	18M	4005	67.1 (+2.2)	74.5 (+2.3)	52.8 (+2.5)	50.1 (+4.1)	22.9 (+2.6)	37.9 (+2.5)
Mixer-B/32-SAM	60M	4209	69.3 (+9.1)	76.4 (+10.2)	54.7 (+9.4)	54.5 (+13.9)	26.3 (+8.0)	43.7 (+8.8)
Mixer-B/16-SAM	59M	1390	73.9 (+11.1)	80.8 (+11.8)	60.2 (+11.9)	59.8 (+17.3)	29.0 (+10.5)	45.9 (+12.5)

2021; Chen et al., 2020; He et al., 2020; Khosla et al., 2020). We couple SAM with the supervised contrastive learning (Khosla et al., 2020) for 350 epochs, followed by fine-tuning the classification head by 90 epochs for both ViT-S/16 and ViT-B/16. We train ViTs under the supervised contrastive learning framework (Khosla et al., 2020). We take the classification token output from the last layer as the encoded representation and retain the structures of the projection and classification heads (Khosla et al., 2020). We employ a batch size 2048 without memory bank (He et al., 2020) and use AutoAugment (Cubuk et al., 2019) with strength 1.0 following Khosla et al. (2020). For the 350-epoch pretraining stage, the contrastive loss temperature is set as 0.1, and we use the LAMB optimizer (You et al., 2020) with learning rate $0.001 \times \frac{\text{batch size}}{256}$ along with a cosine decay schedule. For the second stage, we train the classification head for 90 epochs via a RMSProp optimizer (Tieleman and Hinton, 2012) with base learning rate 0.05 and exponential decay. The weight decays are set as 0.3 and 1e-6 for

Table 5.8: Accuracy on downstream tasks of the models pre-trained on ImageNet. SAM improves ViTs and MLP-Mixers' transferabilities. ViTs transfer better than ResNets of similar sizes.

%	ResNet-50-SAM	ResNet-152-SAM	ViT-S/16	ViT-S/16-SAM	ViT-B/16	ViT-B/16-SAM	Mixer-S/16	Mixer-S/16-SAM	Mixer-B/16	Mixer-B/16-SAM
CIFAR-10	97.4	98.2	97.6	98.2	98.1	98.6	94.1	96.1	95.4	97.8
CIFAR-100	85.2	87.8	85.7	87.6	87.6	89.1	77.9	82.4	80.0	86.4
Flowers	90.0	91.1	86.4	91.5	88.5	91.8	83.3	87.9	82.8	90.0
Pets	91.6	93.3	90.4	92.9	91.9	93.1	86.1	88.7	86.1	92.5
Average	91.1	92.6	90.0	92.6	91.5	93.2	85.4	88.8	86.1	91.7

the first and second stages, respectively. We use a small SAM perturbation strength $\rho = 0.02$.

Compared to the training procedure without SAM, we find considerable performance gain thanks to SAM's smoothing of the contrastive loss geometry, improving the ImageNet top-1 accuracy of ViT-S/16 from 77.0% to 78.1%, and ViT-B/16 from 77.4% to 80.0%. In comparison, the improvement on ResNet-152 is less significant (from 79.7% to 80.0% after using SAM).

5.4.4 When SAM Meets Transfer Learning

We also study the role of smoothed loss geometry in transfer learning. We select four datasets to test ViTs and MLP-Mixers' transferabilities: CIFAR-10/100 (Krizhevsky and Hinton, 2009), Oxford-IIIT Pets (Parkhi et al., 2012b), and Oxford Flowers-102 (Nilsback and Zisserman, 2008). We use image resolution 224 × 224 during fine-tuning on downstream tasks, other settings exactly follow Dosovitskiy et al. (2021a); Tolstikhin et al. (2021) (see Table 5.9). Note that we do not employ SAM during fine-tuning. We perform a grid search over the base learning rates on small sub-splits of the training sets (10% for Flowers and Pets, 2% for CIFAR-10/100). After that, we fine-tune on the entire training sets and report the results on the respective test sets. For comparison, we also include ResNet-50-SAM and

Table 5.9: Hyperparameters for downstream tasks. All models are fine-tuned with 224 × 224 resolution, a batch size of 512, cosine learning rate decay, no weight decay, and grad clipping at global norm 1.

Dataset	**Total steps**	**Warmup steps**	**Base LR**
CIFAR-10	10K	500	{0.001, 0.003, 0.01, 0.03}
CIFAR-100	10K	500	
Flowers	500	100	
Pets	500	100	

ResNet-152-SAM in the experiments. Table 5.8 summarizes the results, which confirm that the enhanced models also perform better after fine-tuning and that MLP-Mixers gain the most from the sharpness-aware optimization.

5.4.5 Varying Weight Decay Strength

Table 5.10: ImageNet accuracy and curvature analysis for ViT-B/16 when we vary the weight decay strength in Adam (AdamW).

Model	**Weight decay**	**ImageNet (%)**	$\|w\|_2$	L_{train}	$L_{train}^{\mathcal{N}}$	λ_{max}
ViT-B/16	0.2	74.2	339.8	0.51	4.22	507.4
	0.3	74.6	269.3	0.65	6.66	738.8
	0.4	74.7	236.7	0.77	7.08	1548.9
	0.5	74.4	211.8	0.98	7.21	2251.7
ViT-B/16-SAM	0.2	79.9	461.4	0.69	0.72	13.1
	0.3	79.9	353.8	0.82	0.96	20.9
	0.4	79.4	301.1	0.85	0.98	26.1
	0.5	78.7	259.6	0.95	1.33	45.5

In this section, we vary the strength of weight decay and see the effects of this commonly used regularization approach. As shown in Table 5.10, weight decay helps improve the accuracy on ImageNet when training without SAM, the weight norm also decreases when we enlarge the decay strength as expected. However, enlarging the weight decay aggravates the problem of converging to a sharper region measured by both $L_{train}^{\mathcal{N}}$ and λ_{max}. Another observation is that $\|w\|_2$ consistently increases after applying SAM for every weight decay strength in Table 5.10, together with the improved ImageNet accuracy and smoother landscape curvature.

CHAPTER 6

Optimizer Search as Symbolic Program Discovery

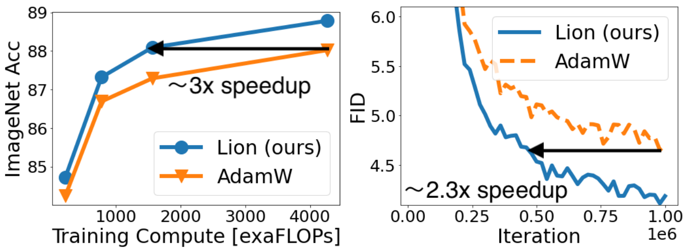
Optimization algorithms, i.e., optimizers, play a fundamental role in training neural networks. There are a large number of handcrafted optimizers, mostly adaptive ones, introduced in recent years (Anil et al., 2020; Balles and Hennig, 2018; Bernstein et al., 2018; Dozat, 2016; Liu et al., 2020; Zhuang et al., 2020). However, Adam (Kingma and Ba, 2014) with decoupled weight decay (Loshchilov and Hutter, 2019), also referred to as AdamW, and Adafactor with factorized second moments (Shazeer and Stern, 2018), are still the de facto standard optimizers for training most deep neural networks, especially the recent state-of-the-art language (Brown et al., 2020; Devlin et al., 2019; Vaswani et al., 2017), vision (Dai et al., 2021; Dosovitskiy et al., 2021b; Zhai et al., 2021) and multimodal (Radford et al., 2021; Saharia et al., 2022; Yu et al., 2022) models.

Another direction is to automatically discover such optimization algorithms. The learning to optimize (L2O) approach Chen et al. (2021a) proposes to discover optimizers by training parameterized models, e.g., neural networks, to output the updates (Andrychowicz et al., 2016; Li and Malik, 2017; Metz et al., 2019, 2022). However, those black-box optimizers, typically trained on a limited number of small tasks, struggle to generalize to state-of-the-art settings where much larger models are trained with significantly more training steps. Another line of methods (Bello et al., 2017; Wang et al., 2022) apply reinforcement learning or Monte Carlo Sampling to discover new optimizers, where the search space is defined by trees composed from predefined operands (e.g., gradient and momentum) and operators (e.g., unary and binary math operations). However, to make the search manageable, they often limit the

Table 6.1: Accuracy of BASIC-L (Pham et al., 2021) on ImageNet and several robustness benchmarks. We apply Lion to both vision tower pre-training and vision-language contrastive training stages. The previous SOTA results on *zero-shot* and *fine-tuning* ImageNet accuracy are 86.3% and 91.0% (Yu et al., 2022).

Optimizer	Zero-shot						Fine-tune
	ImageNet	V2	A	R	Sketch	ObjectNet	ImageNet
Adafactor	85.7	80.6	85.6	95.7	76.1	82.3	90.9
Lion	**88.3**	**81.2**	**86.4**	**96.8**	**77.2**	**82.9**	**91.1**

Figure 6.1: **Left**: ImageNet fine-tuning accuracy vs. pre-training cost of ViT models on JFT-300M. **Right**: FID of the diffusion model on 256^2 image generation. We use DDPM for 1K steps w/o guidance to decode image. As a reference, the FID of ADM is 10.94 (Dhariwal and Nichol, 2021).



Program 6.1: Discovered optimizer Lion. $\beta_1 = 0.9$ and $\beta_2 = 0.99$ by default are derived from Program 6.4. It only tracks momentum and uses the sign operation to compute the update. The two gray lines compute the standard decoupled weight decay, where λ is the strength.

```
def train(weight, gradient, momentum, lr):
  update = interp(gradient, momentum, $\beta_1$)
  update = sign(update)
  momentum = interp(gradient, momentum, $\beta_2$)
  weight_decay = weight * $\lambda$
  update = update + weight_decay
  update = update * lr
  return update, momentum
```

search space by using fixed operands and restricting the size of the tree, thereby limiting the potential for discovery. For example, they are *unable* to modify the tracking of momentum or how it contributes to the update, which is an essential component of Lion. Consequently, the algorithms discovered have not yet reached the state-of-the-art. AutoML-Zero (Real et al., 2020) is an ambitious effort that attempts to search every component of a machine learning pipeline while evaluating on toy tasks. This work follows the research direction of automatic discovering optimizers and is in particular inspired by AutoML-Zero, but aims at discovering effective optimization algorithms that can improve the state-of-the-art benchmarks.

In this chapter, we present a method to formulate algorithm discovery as program search and apply it to discover optimization algorithms. There are two primary challenges. The first one is to find high-quality algorithms in the infinite and sparse program space. The second one is to further select out the algorithms that can generalize from small proxy tasks to much larger, state-of-the-art tasks. To tackle these challenges, we employ a range of techniques including evolutionary search with warm-start and restart, abstract execution, funnel selection, and program simplification.

Our method discovers a simple and effective optimization algorithm: Lion, short for *Evo**L**ved S**i**gn M**o**me**n**tum*. This algorithm differs from various adaptive algorithms by only tracking momentum and leveraging the sign operation to calculate updates, leading to lower memory overhead and uniform update magnitudes across all dimensions. Despite its simplicity, Lion demonstrates outstanding performance across a range of models (Transformer, MLP, ResNet, U-Net, and Hybrid) and tasks (image classification, vision-language contrastive learning, diffusion, language modeling, and fine-tuning). Notably, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet by replacing Adafactor with Lion in BASIC (Pham et al., 2021), surpassing the previous best results by 2% and 0.1%, respectively. Additionally, Lion reduces the pre-training compute on JFT by up to 5x, improves training efficiency on diffusion models by 2.3x and achieves a better FID score, and offers similar or better performance on language modeling with up to 2x compute savings.

We analyze the properties and limitations of Lion. Users should be aware that the uniform update calculated using the sign function usually yields a larger norm compared to those generated by SGD and adaptive methods. Therefore, Lion requires a smaller learning rate lr, and a larger decoupled weight decay λ to maintain the effective weight decay strength. For detailed guidance, please refer to Section 6.4. Additionally, our experiments show that the gain of Lion increases with the batch size and it is more robust to different hyperparameter choices compared to AdamW. For limitations, the difference between Lion and AdamW is not statistical significant on some large-scale language and image-text datasets. The advantage of Lion is smaller if using strong augmentations or a small batch size (<64) during training. See Section 6.5 for details.

6.1 Symbolic Discovery of Algorithms

We present an approach that formulates algorithm discovery as program search (Brameier et al., 2007; Koza, 1994; Real et al., 2020). We use a symbolic representation in the form of programs for the following advantages: (1) it aligns with the fact that algorithms must be implemented as programs for execution; (2) symbolic representations like programs are easier to analyze, comprehend and transfer to new tasks compared to parameterized models such as neural networks; (3) program length can be used to estimate the complexity of different programs, making it easier to select the simpler, often more generalizable ones. This work focuses on optimizers for deep neural network training, but the method is generally applicable to other tasks.

6.1.1 Program Search Space

We adhere to the following three criteria while designing the program search space: (1) the search space should be flexible enough to enable the discovery of novel algorithms; (2) the programs should be easy to analyze and incorporate into a machine learning workflow;

Program 6.2: An example training loop, where the optimization algorithm that we are searching for is encoded within the `train` function. The main inputs are the weight (`w`), gradient (`g`) and learning rate schedule (`lr`). The main output is the `update` to the weight. `v1` and `v2` are two additional variables for collecting historical information.

```
w = weight_initialize()
v1 = zero_initialize()
v2 = zero_initialize()
for i in range(num_train_steps):
  lr = learning_rate_schedule(i)
  g = compute_gradient(w, get_batch(i))
  update, v1, v2 = train(w, g, v1, v2, lr)
  w = w - update
```

Program 6.3: Initial program (AdamW). The bias correction and ϵ are omitted for simplicity.

```
def train(w, g, m, v, lr):
  g2 = square(g)
  m = interp(g, m, 0.9)
  v = interp(g2, v, 0.999)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  wd = w * 0.01
  update = update + wd
  lr = lr * 0.001
  update = update * lr
  return update, m, v
```

Program 6.4: Discovered program after search, selection and removing redundancies in the raw Program 6.8. Some variables are renamed for clarity.

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  g = arcsin(g)
  m = interp(g, v, 0.899)
  m2 = m * m
  v = interp(g, m, 1.109)
  abs_m = sqrt(m2)
  update = m / abs_m
  wd = w * 0.4602
  update = update + wd
  lr = lr * 0.0002
  m = cosh(update)
  update = update * lr
  return update, m, v
```

(3) the programs should focus on the high-level algorithmic design rather than low-level implementation details. We define the programs to contain functions operating over n-dimensional arrays, including structures like lists and dictionaries containing such arrays, in an imperative language. They are similar to Python code using NumPy / JAX (Bradbury et al., 2018; Harris et al., 2020) as well as pseudo code of optimization algorithms. The details of the design are outlined below, with an example representation of AdamW in Program 6.3.

Input / output signature The program defines a `train` function, which encodes the optimization algorithm being searched for, where the main inputs are the model weight (`w`), the gradient (g) and the learning rate schedule value (lr) at the current training step. The main output is the update to the weight. The program also incorporates extra variables initialized as zeros to collect historical information during training. For example, AdamW requires two extra variables to estimate first and second moments. Note that those variables can be used arbitrarily, we use the name m and v in Program 6.3 just for better readability. This simplified code snippet in Program 6.2 uses the same signature as AdamW to ensure that the discovered algorithms have smaller or equal memory footprints. As opposed to previous optimizer search attempts (Bello et al., 2017; Wang et al., 2022), our method allows discovering better ways of updating the extra variables.

Building blocks The train function consists of a sequence of assignment statements, with no restrictions on the number of statements or local variables. Each statement calls a function using constants or existing variables as inputs, and the resulting value is stored in a new or existing variable. For the program, we select 45 common math functions, most of which corresponds to a function in NumPy or an operation in linear algebra. Some functions are introduced to make the program more compact, such as the linear interpolation function interp(x, y, a), which is made equivalent to (1 - a) * x + a * y. Preliminary experiments have investigated the inclusion of more advanced features such as conditional and loop statements, and defining and calling new functions, but these do not yield improved results, so we leave them out. We include 43 available functions that can be used in the program during search. Note that the input of the functions can be one n-dimensional array, dictionaries or lists of arrays, similar to the *pytrees* in JAX.

- **Basic math functions from NumPy / JAX** This includes unary functions like abs, cos, sin, tan, arcsin, arccos, arctan, exp, log, sinh, cosh, tanh, arcsinh, arccosh, arctanh, sign, exp2, exp10, expm1, log10, log2, log1p, square, sqrt, cube, cbrt, sign, reciprocal and binary functions like +, -, *, /, power, maximum, minimum with the same semantic as the corresponding function in NumPy / JAX.

- **Linear algebra functions commonly used in first-order optimization algorithms** This includes: (1) unary function norm that computes the norm of each arrays in the input; (2) unary function global_norm that computes the global norm by treating all the numbers in the input as one vector; (3) binary function dot that treats the two inputs as two vectors and computes their dot product; (4) binary function cosine_sim that treats the two inputs as two vectors and computes their cosine similarity; (5) binary clip_by_global_norm (clip) that clips the global norm of the first input to the value of the second input that is required to be a scalar; (6) ternary function interpolate (interp) that uses the third argument a, required to be a scalar, to compute a linear interpolation of the first two arguments x and y with (1 - a) * x + a * y.

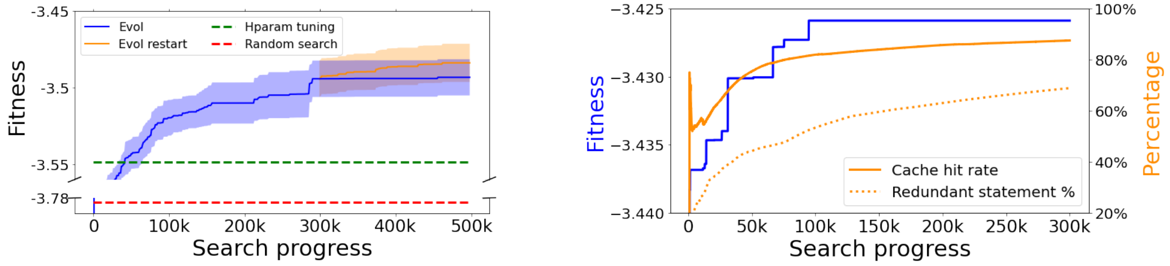
- **Functions producing commonly used constants** This includes get_pi, get_e, get_eps that generates π, e and $\epsilon = 10^{-8}$ respectively.

When necessary, the types and shapes of the function arguments are automatically cast, e.g., in the case of adding a dictionary of arrays to a scalar.

Mutations and redundant statements The design of mutations utilized in evolutionary search is tightly intertwined with the representation of the program. We include three types of mutations: (1) inserting a new statement at a random location with randomly chosen functions and arguments, (2) deleting a random chosen statement, and (3) modifying a random statement by randomly altering one of its function arguments, which may be either variables or constants. To mutate an argument, we replace it with an existing variable or a newly generated constant obtained by sampling from a normal distribution $X \sim \mathcal{N}(0\ 1)$. Additionally, we can mutate an existing constant by multiplying it by a random factor 2^a, where $a \sim \mathcal{N}(0\ 1)$. These constants serve as tunable hyperparameters in the optimization algorithm, such as the peak learning rate and weight decay in AdamW. Note that we allow a program to include redundant statements during search, i.e., statements that do not impact the final program outputs. This is necessary as mutations are limited to only affecting a single statement. Redundant statements therefore serve as intermediate steps towards future substantial modifications in the program.

Infinite and sparse search space Given the limitless number of statements and local variables, as well as the presence of mutable constants, the program search space is infinite. Even if we ignore the constants and bound the program length and number of variables, the number of potential programs is still intractably large. A rough estimate of the number of possible programs is $n_p = n_f^l n_v^{n_a * l}$, where n_f is the number of possible functions, n_v is the number of local variables, n_a is the average number of arguments per statement, and l is the program length. More importantly, the challenge comes from the sparsity of high-performing programs in the search space. To illustrate this point, we conduct a random search that evaluates over 2M programs on a low-cost proxy task. The best program among them is still significantly inferior to AdamW.

Figure 6.2: **Left**: We run hyperparameter tuning on AdamW and random search, both with 4x more compute, to get the best results as two baselines (green and red lines). The evolutionary search, with mean and standard error calculated from five runs, significantly outperforms both of them. The use of multiple restarts from the initial program is crucial due to the high variance in the search fitness (blue curves), and restarting from the best program after 300K progress further improves the fitness (orange curves) when the original search plateaus. **Right**: Example curves of search fitness, the cache hit rate, and the percentage of redundant statements. The cache hit rate and the redundant statements percentage increase along with the search progress to ∼90% and ∼70%.



6.1.2 Efficient Search Techniques

We employ the following techniques to address the challenges posed by the infinite and sparse searching space.

Evolution with warm-start and restart We apply regularized evolution as it is simple, scalable, and has shown success on many AutoML search tasks (Holland, 1992; Real et al., 2019b, 2020; So et al., 2019; Ying et al., 2019). It keeps a population of P algorithms that are gradually improved through cycles. Each cycle picks $T < P$ algorithms at random and the best performer is chosen as the *parent*, i.e., *tournament selection* (Goldberg and Deb, 1991). This parent is then copied and *mutated* to produce a *child* algorithm, which is added to the population, while the oldest algorithm is removed. Normally, evolutionary search starts with random candidates, but we warm-start the initial population as AdamW to accelerate the search. By default, we use a tournament size of two and a population size of 1K. To further improve the search efficiency, we apply two types of restart: (1) restarting from the initial program, which can lead to different local optima due to the randomness in evolution and encourage exploration. This can be done by running multiple searches in parallel. (2) restarting from the best algorithm found thus far to further optimize it, encouraging exploitation. Figure 6.2 (Left) displays the mean and standard error of five evolutionary search experiments. We run hyperparameter tuning based on AdamW by only allowing mutations of constants in the evolution, and run random search by sampling random programs, both with 4x more compute. Our search significantly outperforms the best results achieved by both baselines, demonstrated as the two dashed lines in the figure. The high variance in the search fitness necessitates running multiple repeats through restarting from the initial program. When the search fitness plateaus after ∼300K progress, restarting from the best program found thus far further improves the fitness shown by the orange curve.

Pruning through abstract execution We propose to prune the redundancies in the program space from three sources: programs with syntax or type / shape errors, functionally equivalent programs, and redundant statements in the programs. Before a program is actually executed, we perform an abstract execution step that (1) infers variable types and shapes to detect programs with errors, and keeps mutating the parent program until a valid child program is generated; (2) produces a hash that uniquely identifies how the outputs are computed from the inputs, allowing us to cache and look up semantically duplicate programs (Gillard et al., 2023); (3) identifies redundant statements that can be ignored during actual execution and analysis. For instance, Program 6.4 is obtained after removing all redundant statements in Program 6.8. Abstract execution has negligible cost compared to the actual execution, with each input and function replaced by customized values, e.g., hash. We outline the specifics of the customized values and abstract execution procedure for three use cases below.

- **Detecting errors with type / shape inference** To detect programs containing errors, we infer the type and shape of each variable in the program through the following steps: (1) replace each input with an abstract object that only contains type and shape information, and replace each statement with a type and shape inference function; (2) iterate through all statements. Instead of executing the original statement, we validate a function call by checking the function signature and type and shape information of its arguments. If valid, we compute the type and shape information of the output and assign it to the new variable; (3) verify the validity of the derived type and shape of the output. This process essentially performs a static analysis of the program, exposing errors caused by type and shape mismatch. Note that there are still run-time errors, such as division by zero, that cannot be detected in this manner. Without such filtering of invalid programs, the search would be overwhelmed with invalid programs, making it difficult to achieve meaningful progress.

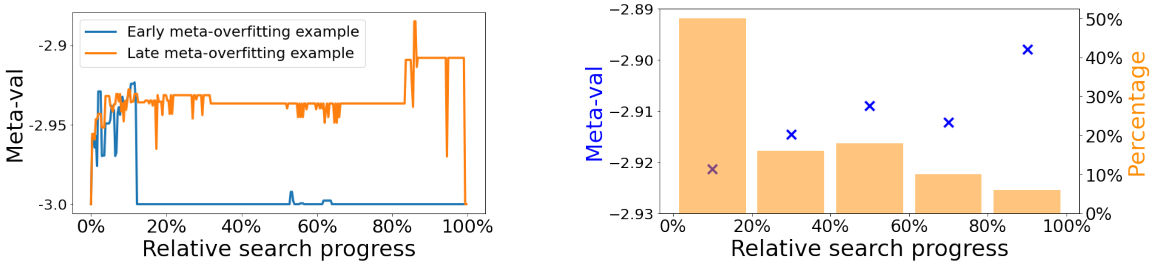
- **Deduplicating with functional hash** Among the valid programs that execute without errors, there are still lots of duplicates due to functionally equivalent programs that have different surface forms but the same underlying functionality. To address this issue, we calculate a functional hash value for every unique computation from the inputs to the outputs as follows: (1) a unique hash value is assigned to each input and function; (2) iterate through all statements, calculating the hash value of the outputs by combining the hash values of the functions and arguments; (3) compute the hash value of program by combining the hash values of all outputs. We then build a hash table that maps each unique functional hash value to the fitness of the corresponding program. When a new program is generated, we first look up its hash value and only perform evaluation if it is not found or if we want to evaluate it multiple times to reduce measurement noise. In our experiments, this technique reduces the search cost by ∼10x, as depicted in Figure 6.2 (Right).

- **Identifying redundant statements by tracking dependencies** In program evolution, redundant statements are included to enable combining multiple mutations to make larger program changes. However, these redundant statements increase the evaluation cost and make program analysis more challenging. To identify redundant statements, we need to determine the set of statements that the outputs depend on, which can be computed in a recursive manner using the following steps: (1) replace the value of each input with an empty set, as they do not depend on any statement; (2) iterate through each statement. Note that each statement is an assignment that calls a function and assigns the result to a variable, which in turn depends on the current statement and all the depending statements of the function arguments. Therefore we replace the value of the variable with its dependency, i.e., a set of all depending statements; (3) compute the union of all statements that each output depends on, which contains all non-redundant statements. By filtering out redundant statements, we obtain a simplified version of the program that is cheaper to execute and easier to analyze. In our experiments, this reduces the program length by ∼3x on average, as shown in Figure 6.2 (Right).

Preliminary experiments have shown that the search process can become overwhelmed with invalid programs and cannot make progress without filtering out invalid programs. As seen in Figure 6.2 (Right), the percentage of redundant statements and cache hit rate both increase as the search proceeds. Based on five search runs, each covering 300K programs, there are $69.8 \pm 1.9\%$ redundant statements towards the end, implying that redundant statements removal makes the program ∼3x shorter on average, thus easier to analyze. The cache hit rate is $89.1 \pm 0.6\%$, indicating that using the hash table as cache brings ∼10x reduction on the search cost.

Proxy tasks and search cost To reduce search cost, we create low-cost proxies by decreasing the model size, number of training examples, and steps from the target tasks. Evaluation on the proxies can be completed on one TPU V2 chip within 20min. We use the accuracy or perplexity on the validation set as the fitness. Each search experiment utilizes 100 TPU V2 chips and runs for ∼72h. There are a total of 200-300K programs generated during each search experiment. However, the number of programs that are actually evaluated is around 20-30K, thanks to the use of the cache through abstract execution. To incorporate restart, we start five repeats of search experiments, followed by another round of search initializing from the best algorithm found thus far. This results in a total cost of ∼3K TPU V2 days. The details of proxy tasks are as follows: For vision tasks, we train a ViT with three layers, 96 hidden units and three heads, on 10% ImageNet for 30k steps with batch size 64. The image size is 64×64 and the patch size is 16. For language tasks, we train a Transformer with two layers, 128 hidden units and two heads on LM1B (Chelba et al., 2013) for 20K steps with batch size 64, sequence length 32 and vocabulary size 3K. The evaluation time may vary for different programs, but typically a evaluation can be done on one TPU V2 chip within 20min. The validation accuracy or perplexity is used as the fitness.

Figure 6.3: **Left**: The meta-validation (defined in Section 6.1.3) curves of two search runs measured on a ∼500x larger meta-validation task compared to the proxy. The blue one meta-overfits at ∼15% of the search progress, while the orange one meta-overfits at ∼90% and achieves a better metric. **Right**: Histogram of the search progress when meta-overfitting happens based on 50 runs. Half of the runs meta-overfit early but a long tail of runs meta-overfit much later. Blue cross depicts the best meta-validation metric averaged within each bin, indicating that meta-overfitting happening later leads to programs that generalize better.



6.1.3 Generalization: Program Selection and Simplification

The search experiments can discover promising programs on proxy tasks. We use performance on *meta-validation* tasks that are larger than the proxy tasks by increasing the model size and training steps, to select the programs that generalize beyond proxy tasks then further simplify them. The phenomenon of *meta-overfitting* occurs when the search fitness keeps growing, but the meta-validation metric declines, indicating that the discovered algorithms have overfit the proxy tasks. Two examples are shown in Figure 6.3 (Left), where the blue curve represents early meta-overfitting and the orange curve represents later meta-overfitting.

Large generalization gap The discovered algorithms face a significant challenge due to the substantial gap between the proxy tasks during search and the target tasks. While proxy tasks can typically be completed within 20min on one TPU V2 chip, target tasks can be $> 10^4$x larger and require days of training on 512 TPU V4 chips. Furthermore, we expect the optimizer to perform well on different architectures, datasets and even different domains, so the discovered algorithms need to show strong out-of-distribution generalization. The sparse search space and inherent noise in the evolution process further compound this challenge, leading to inconsistent generalization properties between different runs. Our observation suggests that evolutionary search experiments that meta-overfit later tend to uncover optimization algorithms that generalize better. See more details in Figure 6.3 (Right).

Funnel selection To mitigate the generalization gap, we collect promising programs based on search fitness and add an extra selection step using a series of meta-validation tasks to select those generalize better. To save compute, we apply a funnel selection process that gradually increases the scale of the meta-validation tasks. For example, starting with proxy task A, we create a 10x larger task B by increasing the model size and the training steps. Only algorithms that surpass the baseline on task B will be evaluated on task C, which is 100x larger. This approach allows us to gradually filter out algorithms that show poor generalization performance, ultimately leading to the selection of algorithms that generalize well to larger tasks.

Simplification Simpler programs are easier to understand and our intuition is that they are more likely to generalize, so we simplify the programs with the following steps. Firstly, we remove redundant statements that do not contribute to the final output as identified through abstract execution. Secondly, we remove statements that are non-redundant but produce minimal differences when removed. This step can also be achieved through evolution by disabling the insertion of new statements in the mutation process. Finally, we rearrange the statements manually, assign clear and descriptive names to variables, and convert the program into its simpler, mathematically equivalent form.

Program 6.5: Algorithm with a better regularization. It dynamically calculates the dot product between the weight and gradient, before computing the weight decay.

```
def train(w, g, m, v, lr):
  m = interp(m, g, 0.16)
  g2 = square(g)
  v = interpolate(v, g2, 0.001)
  v753 = dot(g, w)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  wd = v753 * w
  update = sin(update)
  update = update + wd
  lr = lr * 0.0216
  update = update * lr
  v = sin(v)
  return update, m, v
```

Program 6.6: Algorithm that tracks the second moment without EMA decay similar to AdaGrad.

```
def train(w, g, m, v, lr):
  m = interp(m, g, 0.1)
  g2 = square(g)
  g2 = v + g2
  v = interp(v, g2, 0.0015)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  v70 = get_pi()
  v = min(v, v70)
  update = sinh(update)
  lr = lr * 0.0606
  update = update * lr
  return update, m, v
```

Program 6.7: Algorithm uses the difference between gradient and momentum to track the second moment, resembling AdaBelief.

```
def train(w, g, m, v, lr):
  m = interp(m, g, 0.1)
  g = g - m
  g2 = square(g)
  v = interp(v, g2, 0.001)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  wd = w * 0.0238
  update = update + wd
  lr = lr * 0.03721
  update = update * lr
  return update, m, v
```

6.2 Derivation and Analysis of Lion

We arrive at the optimizer Lion due to its simplicity, memory efficiency, and strong performance in search and meta-validation. Note that the search also discovers other existing or novel algorithms. By varying the task setting, different types of algorithms can be discovered. For example, if we reduce the amount of data in the proxy task, we are more likely to discover algorithms with better regularization (Program 6.5), and if we reduce the search progress, we are likely to find simple variants of AdamW (Program 6.6 and 6.7). Future work can explore this potential to discover optimizers specialized for different tasks.

Program 6.8: Raw program of Lion before removing redundent statements.

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  m = clip(m, lr)
  v845 = sqrt(0.6270633339881897)
  v968 = sign(v)
  v968 = v - v
  g = arcsin(g)
  m = interp(g, v, 0.8999999761581421)
  v1 = m * m
  v = interp(g, m, 1.109133005142212)
  v845 = tanh(v845)
  lr = lr * 0.0002171761734643951
  update = m * lr
  v1 = sqrt(v1)
  update = update / v1
  wd = lr * 0.4601978361606598
  v1 = square(v1)
  wd = wd * w
  m = cosh(update)
  lr = tan(1.4572199583053589)
  update = update + wd
  lr = cos(v845)
  return update, m, v
```

6.2.1 Derivation

The search and funnel selection process lead to Program 6.4, which is obtained by automatically removing redundant statements from the raw Program 6.8. We further simplify it to get the final algorithm (Lion) in Program 6.1. Several unnecessary elements are removed from

Program 6.4 during the simplification process. The `cosh` function is removed since `m` would be reassigned in the next iteration (line 3). The statements using `arcsin` and `clip` are also removed as we observe no quality drop without them. The three orange statements translate to a single `sign` function. Although both `m` and `v` are utilized in Program 6.4, `v` only changes how the momentum is updated (two `interp` functions with constants ∼0.9 and ∼1.1 is equivalent to one with ∼0.99) and does not need to be separately tracked. Note that the bias correction is no longer needed, as it does not change the direction. Algorithm 4 shows the pseudocode.

Algorithm 3 AdamW Optimizer

```
given $\beta_1$, $\beta_2$, $\epsilon$, $\lambda$, $\eta$, $f$
initialize $\theta_0$, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    update EMA of $g_t$ and $g_t^2$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    bias correction
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
    update model parameters
    $\theta_t \leftarrow \theta_{t-1} - \eta_t (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1})$
end while
return $\theta_t$
```

Algorithm 4 Lion Optimizer (ours)

```
given $\beta_1$, $\beta_2$, $\lambda$, $\eta$, $f$
initialize $\theta_0$, $m_0 \leftarrow 0$
while $\theta_t$ not converged do
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    update model parameters
    $c_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $\theta_t \leftarrow \theta_{t-1} - \eta_t (\text{sign}(c_t) + \lambda \theta_{t-1})$
    update EMA of $g_t$
    $m_t \leftarrow \beta_2 m_{t-1} + (1 - \beta_2) g_t$
end while
return $\theta_t$
```

6.2.2 Analysis

Sign update and regularization The Lion algorithm produces update with uniform magnitude across all dimensions by taking the sign operation, which is in principle different from various adaptive optimizers. Intuitively, the sign operation adds noise to the updates, which acts as a form of regularization and helps with generalization (Chen et al., 2022; Foret et al., 2021a; Neelakantan et al., 2017). An evidence is shown in Figure 6.9 (Right), where the ViT-B/16 trained by Lion on ImageNet has a higher training error compared to AdamW but a 2% higher accuracy on the validation set (as shown in Table 6.2). Additionally, the results in Section 6.3.6 demonstrate that Lion leads to the convergence in smoother regions, which usually results in better generalization.

Momentum tracking The default EMA factor used to track the momentum in Lion is 0.99 (β_2), compared to the commonly used 0.9 in AdamW and momentum SGD. The current gradient and momentum are interpolated with a factor of 0.9 (β_1) before the sign operation is applied. This choice of EMA factor and interpolation allows Lion to balance between remembering a $\sim$10x longer history of the gradient in momentum and putting more weight on the current gradient in the update. The necessity of both β_1 and β_2 is further discussed in Section 6.3.6.

Hyperparameter and batch size choices Lion is simpler and has fewer hyperparameters compared to AdamW and Adafactor as it does not require ϵ and factorization-related ones. The update is an element-wise binary ±1 if we omit the weight decay term, with larger norm than those produced by other optimizers like SGD and adaptive algorithms. As a result, Lion needs a *smaller* learning rate and in turn a *larger* decoupled weight decay to achieve a similar effective weight decay strength (`lr *` λ). Detailed information on tuning Lion can be found in Section 6.4. Additionally, the advantage of Lion over AdamW enlarges as the batch size increases, which fits the common practice of scaling up model training through data parallelism (Section 6.3.6).

Memory and runtime benefits Lion only saves the momentum thus has smaller memory footprint than popular adaptive optimizers like AdamW, which is beneficial when training large models and / or using a large batch size. As an example, AdamW needs at least 16 TPU V4 chips to train a ViT-B/16 with image resolution 224 and batch size 4,096, while Lion only needs 8 (both with `bfloat16` momentum). Another practical benefit is that Lion has faster runtime (steps / sec) in our experiments due to its simplicity, usually 2-15% speedup compared to AdamW and Adafactor depending on the task, codebase, and hardware.

Relation to existing optimizers The sign operation has been explored in previous optimizers (Bernstein et al., 2018; Riedmiller and Braun, 1993). The closest to ours is the handcrafted optimizer signSGD (Bernstein et al., 2018) (and its momentum variant) that also utilizes the sign operation to calculate the update but has a different momentum update rule from Lion. Their focus is to mitigate communication costs between agents in distributed training, and they observe inferior performance when training ConvNets on image classification tasks. On the other hand, NAdam (Dozat, 2016) combines the updated first moment and the gradient to compute the update, but Lion decouples the momentum tracking and how it is applied to the update through β_2. A comparison of Lion with related optimizers can be found in Section 6.3.5.

6.3 Evaluation of Lion

In this section, we present evaluations of Lion, on various benchmarks. We mainly compare it to AdamW (or Adafactor when memory is a bottleneck) as it is exceedingly popular and the de facto standard optimizer on a majority of learning tasks. The result of momentum SGD is only included for ResNet since it performs worse than AdamW elsewhere. We also benchmark other popular optimizers in Section 6.3.5, including handcrafted and automatically discovered ones. We make sure that every optimizer is well-tuned for each task (see Section 6.4 for tuning details). By default, the learning rate schedule is cosine decay with 10K steps warmup, and the momentum is saved as `bfloat16` to reduce the memory footprint.

6.3.1 Image Classification

We perform experiments including various datasets and architectures on the image classification task. Apart from training from scratch on ImageNet, we also pre-train on two larger well-established datasets, ImageNet-21K and JFT (Sun et al., 2017a). The image size is 224^2 by default otherwise specified by the subscript. Our evaluation covers various benchmarks: ImageNet, ImageNet ReaL (Beyer et al., 2020b), ImageNet V2 (Recht et al., 2019b), ImageNet A (Hendrycks et al., 2021b), ImageNet R (Hendrycks et al., 2021a), ImageNet Sketch (Wang et al., 2019b), ObjectNet (Barbu et al., 2019), CIFAR-100 (Krizhevsky, 2009), and Oxford-IIIT Pet (Parkhi et al., 2012a).

Train from scratch on ImageNet Following previous works (Dosovitskiy et al., 2021b; He et al., 2016), we train ResNet-50 for 90 epochs with a batch size of 1,024, and other models for 300 epochs with a batch size of 4,096. As shown in Table 6.2, Lion significantly outperforms AdamW on various architectures. Empirically, the improvement is more substantial on models with larger capacity, with accuracy increases of 1.96% and 0.58% for ViT-B/16 and ViT-S/16, respectively. The performance gaps also tend to enlarger with fewer inductive biases. When strong augmentations are applied, the gain of Lion over AdamW shrinks, but it still outperforms AdamW by 0.42% on CoAtNet-3, despite the strong regularization during training (Dai et al., 2021).

Pre-train on ImageNet-21K We pre-train ViT-B/16 and ViT-L/16 on ImageNet-21K for 90 epochs with a batch size of 4,096. Table 6.2 shows that Lion still surpasses AdamW even when the training set is enlarged for 10x. The gaps on larger models are consistently

Table 6.2: Accuracy on ImageNet, ImageNet ReaL, and ImageNet V2. Numbers in (·) are from Dai et al. (2021); Dosovitskiy et al. (2021b). Results are averaged from three runs.

Model	#Params	Optimizer	RandAug + Mixup	ImageNet	ReaL	V2
Train from scratch on ImageNet						
ResNet-50	25.56M	SGD	✗	76.22	82.39	63.93
		AdamW		76.34	**82.72**	**64.24**
		Lion		**76.45**	**82.72**	64.02
Mixer-S/16	18.53M	AdamW	✗	69.26	75.71	55.01
		Lion		**69.92**	**76.19**	**55.75**
Mixer-B/16	59.88M	AdamW	✗	68.12	73.92	53.37
		Lion		**70.11**	**76.60**	**55.94**
ViT-S/16	22.05M	AdamW	✗	76.12	81.94	63.09
		Lion		**76.70**	**82.64**	**64.14**
		AdamW	✓	78.89	84.61	66.73
		Lion		**79.46**	**85.25**	**67.68**
ViT-B/16	86.57M	AdamW	✗	75.48	80.64	61.87
		Lion		**77.44**	**82.57**	**64.81**
		AdamW	✓	80.12	85.46	68.14
		Lion		**80.77**	**86.15**	**69.19**
CoAtNet-1	42.23M	AdamW	✓	83.36 (83.3)	-	-
		Lion		**84.07**	-	-
CoAtNet-3	166.97M	AdamW	✓	84.45 (84.5)	-	-
		Lion		**84.87**	-	-
Pre-train on ImageNet-21K then fine-tune on ImageNet						
ViT-B/16_{384}	86.86M	AdamW	✗	84.12 (83.97)	88.61 (88.35)	73.81
		Lion		**84.45**	**88.84**	**74.06**
ViT-L/16_{384}	304.72M	AdamW	✗	85.07 (85.15)	88.78 (88.40)	75.10
		Lion		**85.59**	**89.35**	**75.84**

Table 6.3: Model performance when pre-trained on JFT then fine-tuned on ImageNet. Two giant ViT models are pre-trained on JFT-3B while smaller ones are pre-trained on JFT-300M. The ViT-G/14 results are directly from Zhai et al. (2021).

Model	ViT-L/16$_{512}$		ViT-H/14$_{518}$		ViT-g/14$_{518}$		ViT-G/14$_{518}$	
#Params	305.18M		633.47M		1.04B		1.88B	
Optimizer	AdamW	Lion	AdamW	Lion	Adafactor	Lion	Adafactor	Lion
ImageNet	87.72	**88.50**	88.55	**89.09**	90.25	**90.52**	90.45	**90.71** / **90.71***
ReaL	90.46	**90.91**	90.62	**91.02**	90.84	**91.11**	90.81	**91.06** / **91.25***
V2	79.80	**81.13**	81.12	**82.24**	83.10	**83.39**	83.33	**83.54** / **83.83***
A	52.72	**58.80**	60.64	**63.78**	-	-	-	-
R	66.95	**72.49**	72.30	**75.07**	-	-	-	-

* We observe overfitting in fine-tuning, therefore report both the last and oracle results.

bigger, with +0.52% vs. +0.33% (ImageNet), +0.57% vs. +0.23% (ReaL), and +0.74% vs. +0.25% (V2) for ViT-L/16 and ViT-B/16, respectively.

Pre-train on JFT To push the limit, we conduct extensive experiments on JFT. We follow the settings of Dosovitskiy et al. (2021b) and Zhai et al. (2021) for both pre-training and fine-tuning. Figure 6.1 (Left) and 6.4 present the accuracy of three ViT models (ViT-B/16, ViT-L/16, and ViT-H/14) under different pre-training budgets on JFT-300M. Lion enables the ViT-L/16 to match the performance of ViT-H/14 trained by AdamW on ImageNet and ImageNet V2 but with 3x less pre-training cost. On ImageNet ReaL, the compute saving further becomes 5x. Another evidence is that even when a ViT-L/16 is trained by AdamW for 4M steps by Zhai et al. (2021), its performance still lags behind the same model trained by Lion for 1M steps.

Table 6.3 shows the fine-tuning results, with higher resolution and Polyak averaging. Our ViT-L/16 matches the previous ViT-H/14 results trained by AdamW, while being 2x smaller. The advantage is larger on more challenging benchmarks, such as +1.33% (V2), +6.08% (A), +5.54% (R) for ViT-L/16. After we scale up the pre-training dataset to JFT-3B, the

Figure 6.4: ImageNet ReaL (**Left**) and ImageNet V2 (**Right**) accuracy after we pre-train ViT models on JFT-300M then fine-tune on ImageNet.

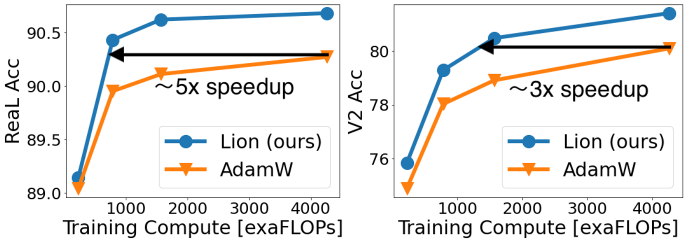


Table 6.4: Zero-shot accuracy of LiTs on ImageNet, CIFAR-100, and Oxford-IIIT Pet. As a reference, the zero-shot accuracy of CLIP (Radford et al., 2021) on ImageNet is 76.2%.

Model	Optimizer	ImageNet	C100	Pet
LiT-B/32-B	AdamW	68.78	71.41	86.62
	Lion	**69.88**	**71.78**	**87.36**
LiT-B/16-B	AdamW	74.26	72.25	89.83
	Lion	**75.39**	**72.49**	**91.20**
LiT-g/14$_{288}$-L	AdamW	83.43	80.93	94.88
	Lion	**84.09**	**81.43**	**95.86**

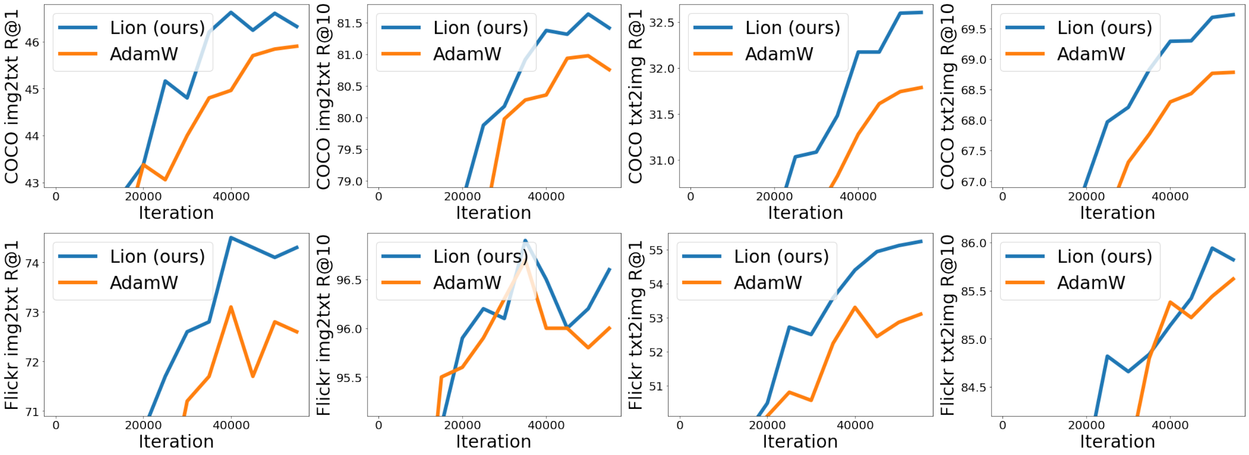
ViT-g/14 trained by Lion outperforms the previous ViT-G/14 results (Zhai et al., 2021), with 1.8x fewer parameters. Our ViT-G/14 further achieves a 90.71% accuracy on ImageNet.

6.3.2 Vision-Language Contrastive Learning

This section focuses on the vision-language contrastive training (Radford et al., 2021). We compare Lion with AdamW (Adafactor) on zero-shot image classification and image-text retrieval benchmarks. Instead of learning all the parameters from scratch, we initialize the image encoder with a strong pre-trained model as it is suggested to be more efficient (Zhai et al., 2022).

Locked-image text Tuning (LiT) We perform a comparison between Lion and AdamW on LiT (Zhai et al., 2022) by training the text encoder (Zhai et al., 2022) in a contrastive manner using the same frozen pre-trained ViT. All models are trained for 1B image-text pairs with a batch size of 16,384. Table 6.4 shows the zero-shot image classification results on three model scales, with the name specifies the size, e.g., LiT-B/16-B denotes a ViT-B/16

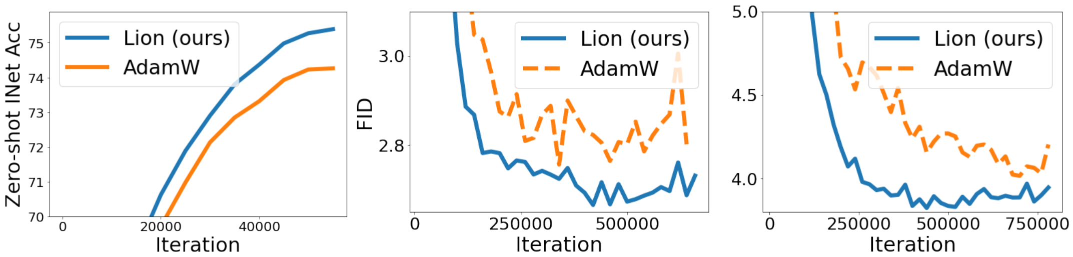
Figure 6.5: Zero-shot image-text retrieval results on MSCOCO (**Top**) and Flickr30K (**Bottom**) for LiT-B/16-B. Recall@K is calculated based on if the ground truth label of the query appears in the top-K retrieved examples.



and a base size Transformer as the text encoder. Our method, Lion, demonstrates consistent improvement over AdamW with gains of +1.10%, +1.13%, and +0.66% on zero-shot ImageNet accuracy for LiT-B/32-B, LiT-B/16-B, and LiT-g/14_{288}-L, respectively. Figure 6.6 (Left) depicts an example zero-shot learning curve of LiT-B/16-B. Similar results are obtained on the other two datasets. The zero-shot image-text retrieval results on MSCOCO (Lin et al., 2014) and Flickr30K (Plummer et al., 2015) can be found in Figure 6.5. The evaluation metric is Recall@K, calculated based on if the ground truth label of the query appears in the top-K retrieved examples. Lion outperforms AdamW on both datasets, with a larger gain in Recall@1 than Recall@10 on Flicker30K, implying more accurate retrieval results: +1.70% vs. +0.60% for image → text and +2.14% vs. +0.20% for text → image.

BASIC Pham et al. (2021) propose to scale up batch size, dataset, and model size simultaneously, achieving drastic improvements over CLIP. It uses a sophisticated CoAtNet (Dai et al., 2021) pre-trained on JFT-5B as the image encoder. Furthermore, the contrastive training is

Figure 6.6: The zero-shot ImageNet accuracy curve of LiT-B/16-B (**Left**). FID comparison on 64×64 (**Middle**) and 128×128 (**Right**) image generation when training diffusion models. We decode image w/o guidance.



performed on 6.6B image-text pairs with a larger 65,536 batch size. To push the limit, we only experiment on the largest BASIC-L, and use Lion on *both* image encoder pre-training and contrastive learning stages. As illustrated in Table 6.1, we achieve a significant 2.6% gain over the baseline, striking a 88.3% accuracy on zero-shot ImageNet classification. Note that this result is 2.0% higher than the previous best result (Yu et al., 2022). The performance gain is consistent on five other robustness benchmarks. After fine-tuning the image encoder (CoAtNet-7) in BASIC-L obtained by Lion, we further achieve a 91.1% top-1 accuracy on ImageNet, which is 0.1% better than the previous SOTA.

6.3.3 Diffusion Model

Recently, diffusion models achieve a huge success on image generation (Dhariwal and Nichol, 2021; Ho and Salimans, 2022; Ho et al., 2020; Saharia et al., 2022; Song et al., 2021). Given its enormous potential, we test the performance of Lion on unconditional image synthesis and multimodal text-to-image generation.

Image synthesis on ImageNet We utilize the improved U-Net architecture introduced in Dhariwal and Nichol (2021) and perform 64×64, 128×128, and 256×256 image generation on ImageNet. The batch size is set as 2,048 and the learning rate remains constant throughout training. For decoding, we apply DDPM (Ho et al., 2020) for 1K sampling steps *without* classifier-free guidance.The evaluation metric is the standard FID score. Illustrated by Figure 6.1 (Right) and 6.6 (Middle and Right), Lion enables both better quality and faster convergence on the FID score. Note that the gap between Lion and AdamW tends to increase with the image resolution, where the generation task becomes more challenging. When generating 256 × 256 images, Lion achieves the final performance of AdamW at 440K steps, reducing 2.3x iterations. The final FID scores are 4.1 (Lion) vs. 4.7 (AdamW), and for reference, the FID of ADM (Dhariwal and Nichol, 2021) is 10.94.

Text-to-image generation We follow the Imagen (Saharia et al., 2022) setup to train a base 64 × 64 text-to-image model and a 64 × 64 → 256 × 256 super-resolution model. All models are trained on a high-quality internal image-text dataset with a batch size of 2,048 and a constant learning rate. Due to computational constraints, our base U-Net has a width of 192 compared to 512 in the original 2B model, while the 600M super-resolution model is identical to the original Imagen setup. Along with the training, 2K images are sampled from the MSCOCO (Lin et al., 2014) validation set for real-time evaluation. We use the CLIP score to measure image-text alignment and the zero-shot FID-30K to measure image fidelity. Classifier-free guidance (Ho and Salimans, 2022) with a weight of 5.0 is applied as it has been shown to improve image-text alignment. Figure 6.7 depicts the learning curve. While there is no clear improvement on the base 64 × 64 model, Lion outperforms AdamW on the text-conditional super-resolution model. It achieves a higher CLIP score and has a less noisy FID metric compared to AdamW.

6.3.4 Language Modeling and Fine-tuning

This section focuses on language modeling and fine-tuning. On language-only tasks, we find that tuning β_1 and β_2 can improve the quality for both AdamW and Lion. See Section 6.4 for tuning details.

Figure 6.7: Evaluation of the Imagen text-to-image 64^2 (**Left**) and the $64^2 \to 256^2$ diffusion models (**Right**).

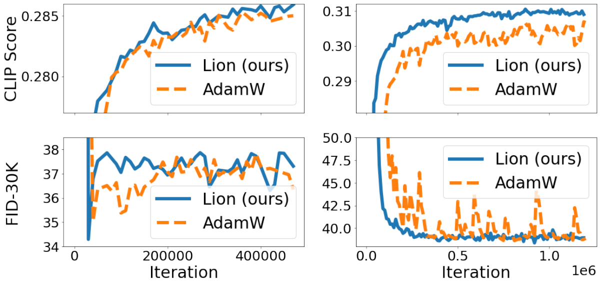
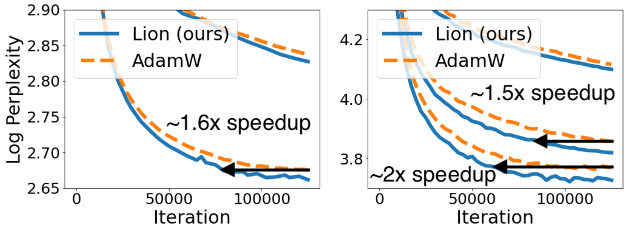


Figure 6.8: Log perplexity on Wiki-40B (**Left**) and PG-19 (**Right**). The speedup brought by Lion tends to increase with the model scale. The largest model on Wiki-40B is omitted as we observe severe overfitting.



Autoregressive language modeling We first experiment on two smaller-scale academic datasets Wiki-40B (Guo et al., 2020) and PG-19 (Rae et al., 2020) following Hua et al. (2022). The employed Transformer spans three scales: small (110M), medium (336M), and large (731M). Table 6.9 shows the Transformer architecture details. The dimension of the feed-forward layer is $4 \times d_{model}$. We use vocabulary size 32K for small-scale and 256K for large-scale models.

All models are trained with 2^{18} tokens per batch for 125K steps, with a learning rate schedule of 10K steps warmup followed by linear decay. The context length is set to 512 for Wiki-40B and 1,024 for PG-19. Figure 6.8 illustrates the token-level perplexity for Wiki-40B and word-level perplexity for PG-19. Lion consistently achieves lower validation perplexity than AdamW. It achieves 1.6x and 1.5x speedup when training the medium size model on Wiki-40B and PG-19, respectively. When the model is increased to the large size, the speedup on PG-19 further increases to 2x.

Scaling up the scale of language models and pre-training datasets has revolutionized the field of NLP. So we further perform larger-scale experiments. Our pre-training dataset, similar to that used in GLaM (Du et al., 2022), consists of 1.6 trillion tokens spanning a wide range

Table 6.5: One-shot evaluation averaged over three NLG and 21 NLU tasks. The results of GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2022) are included for reference. The LLMs trained by Lion have better in-context learning ability. See Table 6.6 for detailed results on all tasks.

Task	1.1B		2.1B		7.5B		6.7B	8B
	Adafactor	Lion	Adafactor	Lion	Adafactor	Lion	GPT-3	PaLM
#Tokens	300B						300B	780B
Avg NLG	11.1	**12.1**	15.6	**16.5**	24.1	**24.7**	23.1	23.9
Avg NLU	53.2	**53.9**	56.8	**57.4**	61.3	**61.7**	58.5	59.4

of natural language use cases. Following GPT-3 (Brown et al., 2020), we train three models, ranging from 1.1B to 7.5B parameters, for 300B tokens with a batch size of 3M tokens and a context length of 1K. We evaluate them on three natural language generative (NLG) and 21 natural language understanding (NLU) tasks. Those tasks include *Open-Domain Question Answering*, *Cloze and Completion Tasks*, *Winograd-Style Tasks*, *Common Sense Reasoning*, *In-Context Reading Comprehension*, *SuperGLUE*, and *Natural Language Inference*.

- NLG: TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019), Web Questions (Berant et al., 2013).

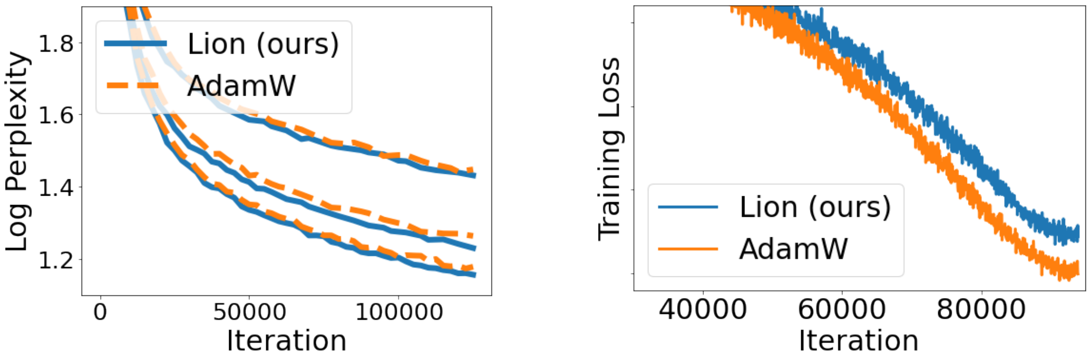
- NLU: HellaSwag (Zellers et al., 2019), StoryCloze (Mostafazadeh et al., 2016), Winograd (Levesque et al., 2012), Winogrande (Sakaguchi et al., 2020), RACE (Lai et al., 2017), PIQA (Bisk et al., 2020), ARC (Clark et al., 2018), OpenbookQA (Mihaylov et al., 2018), BoolQ (Clark et al., 2019), Copa (Gordon et al., 2012), RTE (Dagan et al., 2006), WiC (Pilehvar and Camacho-Collados, 2019), Multirc (Khashabi et al., 2018), WSC (Levesque et al., 2012), ReCoRD (Zhang et al., 2018b), CB (de Marneffe et al., 2019), Adversarial NLI (Nie et al., 2020).

On this massive dataset, we observe no perplexity difference throughout training. Nev-

Table 6.6: One-shot evaluation on English NLP tasks. TriviaQA, NQs, and WebQs are NLG tasks and the rest are NLU tasks.

Task	1.1B		2.1B		7.5B		6.7B	8B
	Adafactor	Lion	Adafactor	Lion	Adafactor	Lion	GPT-3	PaLM
#Tokens	300B						300B	780B
TriviaQA (EM)	21.5	**25.1**	32.0	**33.4**	47.9	**48.8**	44.4	48.5
NQs (EM)	4.3	**4.8**	6.3	**7.3**	**12.3**	12.1	9.8	10.6
WebQs (EM)	**7.5**	6.3	8.4	**8.7**	12.1	**13.3**	15.1	12.6
HellaSwag	**50.7**	50.3	**59.4**	59.3	68.2	**68.3**	66.5	68.2
StoryCloze	**74.8**	74.4	78.2	**78.3**	81.2	**81.5**	78.7	78.7
Winograd	75.1	**80.2**	81.3	**82.1**	**85.3**	84.2	84.6	85.3
Winogrande	59.7	**60.5**	64.8	**65.7**	**71.4**	71.0	65.8	68.3
RACE-m	**52.0**	50.8	**55.1**	53.8	59.1	**61.3**	54.7	57.7
RACE-h	**36.8**	35.4	40.3	**40.7**	**44.5**	43.9	44.3	41.6
PIQA	69.4	**69.9**	71.3	**72.1**	**75.5**	74.5	76.3	76.1
ARC-e	**64.3**	62.0	**69.5**	68.9	72.4	**72.7**	62.6	71.3
ARC-c	31.2	**32.9**	37.3	**38.0**	**43.3**	42.6	41.5	42.3
OpenbookQA	44.8	**48.0**	48.4	**49.0**	51.4	**52.4**	53.0	47.4
BoolQ	54.3	**56.7**	**64.1**	62.9	73.5	**73.9**	68.7	64.7
Copa	75.0	**78.0**	83.0	**84.0**	85.0	**87.0**	82.0	82.0
RTE	**55.6**	52.4	49.8	**59.2**	**63.9**	62.5	54.9	57.8
WiC	**47.6**	47.3	46.1	**48.1**	**50.9**	48.1	50.3	47.3
Multirc (F1a)	35.9	**44.3**	45.0	**48.8**	44.7	**59.2**	64.5	50.6
WSC	**76.5**	75.4	**79.6**	79.3	**86.7**	85.6	60.6	81.4
ReCoRD	73.4	**73.7**	**77.8**	77.7	81.0	**81.1**	88.0	87.8
CB	**46.4**	44.6	**48.2**	44.6	**51.8**	46.4	33.9	41.1
ANLI R1	**33.3**	30.1	**32.4**	31.2	31.5	**34.0**	31.6	32.4
ANLI R2	29.8	**31.8**	29.8	**30.6**	**32.4**	31.9	33.9	31.4
ANLI R3	29.8	**31.8**	31.4	**31.9**	33.6	**34.2**	33.1	34.5
Avg NLG	11.1	**12.1**	15.6	**16.5**	24.1	**24.7**	23.1	23.9
Avg NLU	53.2	**53.9**	56.8	**57.4**	61.3	**61.7**	58.5	59.4

Figure 6.9: **Left**: Validation perplexity when we perform masked language modeling on the C4 dataset. **Right**: Training loss of ViT-B/16 on ImageNet.



ertheless, Lion outperforms Adafactor on the average in-context learning ability, as shown in Table 6.5. Our 7.5B baseline model, trained for 300B tokens, outperforms the 8B PaLM, trained for 780B tokens, demonstrating the strength of our setup. Lion outperforms Adafactor on both NLG and NLU tasks, particularly on the NLG tasks, with an exact match improvement of +1.0, +0.9, and +0.6 for the 1.1B, 2.1B, and 7.5B models, respectively.

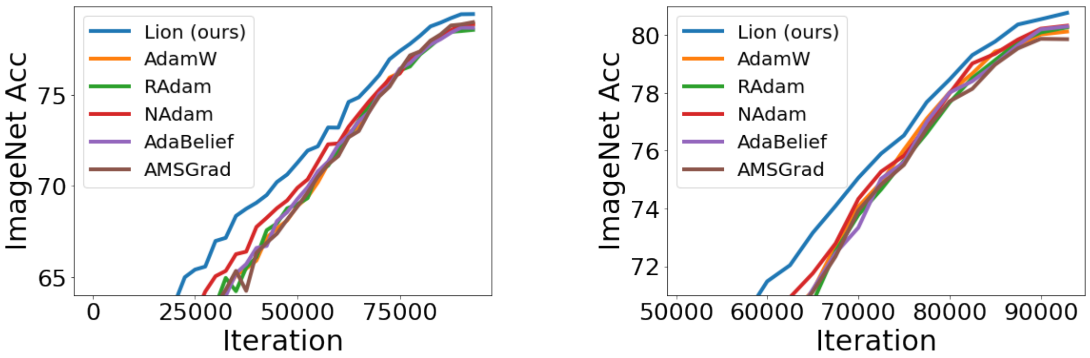
Masked language modeling We also perform BERT training on the C4 dataset (Raffel et al., 2020). It requires the language models to reconstruct randomly masked out tokens in the input sequence. We use the same architectures and training setups as the smaller-scale autoregressive experiments. Lion performs slightly better than AdamW regarding the validation perplexity: 4.18 vs. 4.25 (small), 3.42 vs. 3.54 (medium), and 3.18 vs. 3.25 (large). See Figure 6.9 (Left) for the learning curves.

Fine-tuning We fine-tune Base (220M), Large (770M), and the largest 11B T5 (Raffel et al., 2020) models on the GLUE benchmark (Wang et al., 2019a). Every model is fine-tuned for 500K steps with a batch size of 128 and a constant learning rate. Table 6.7 shows the results on the GLUE dev set. For MRPC and QQP, we report the F1 / Accuracy scores, for STS-B, we report the Pearson / Spearman correlation, and for the other datasets, we report

Table 6.7: Fine-tuning performance of the T5 Base, Large, and 11B on the GLUE dev set. Results reported are the peak validation scores per task.

Model	Optimizer	CoLA	SST-2	MRPC	STS-B	QQP	MNLI -m	MNLI -mm	QNLI	RTE	Avg
Base	AdamW	60.87	95.18	92.39 / 89.22	**90.70** / **90.51**	89.23 / 92.00	86.77	86.91	93.70	81.59	87.42
	Lion	**61.07**	**95.18**	**92.52** / **89.46**	90.61 / 90.40	**89.52** / **92.20**	**87.27**	**87.25**	**93.85**	**85.56**	**87.91**
Large	AdamW	63.89	96.10	93.50 / 90.93	91.69 / 91.56	90.08 / 92.57	89.69	89.92	94.45	89.17	89.46
	Lion	**65.12**	**96.22**	**94.06** / **91.67**	**91.79** / **91.60**	**90.23** / **92.67**	**89.85**	**89.94**	**94.89**	**90.25**	**89.86**
11B	AdamW	69.50	97.02	93.75 / 91.18	92.57 / 92.61	90.45 / 92.85	**92.17**	**91.99**	96.41	92.42	91.08
	Lion	**71.31**	**97.13**	**94.58** / **92.65**	**93.04** / **93.04**	**90.57** / **92.95**	91.88	91.65	**96.56**	**93.86**	**91.60**

Figure 6.10: Learning curve of ViT-S/16 (**Left**) and ViT-B/16 (**Right**) associated with Table 6.8. The curves of the five adaptive optimizers are similar to each other.



their default metric. On average, Lion beats AdamW across all three model scales. It achieves 10, 12, and 10 wins out of 12 scores for T5 Base, Large, and 11B models, respectively.

6.3.5 Comparison with Other Popular Optimizers

We also employ four popular handcrafted optimizers: AdaBelief (Zhuang et al., 2020), AMSGrad (Reddi et al., 2018), RAdam (Liu et al., 2020), NAdam (Dozat, 2016), and two optimizers discovered by AutoML: PowerSign (Bello et al., 2017) and AddSign (Bello et al., 2017) to train ViT-S/16 and ViT-B/16 on ImageNet (with RandAug and Mixup). We

Table 6.8: The performance of various optimizers to train ViT-S/16 and ViT-B/16 on ImageNet (with RandAug and Mixup). Lion is still the best performing one, and there is no clear winner amongst the baselines.

Model	Task	AdamW	RAdam	NAdam	Ada-Belief	AMSGrad	Power-Sign	Add-Sign	Ablation$_{0.9}$	Ablation$_{0.99}$	Lion
ViT-S/16	ImageNet	78.89	78.59	78.91	78.71	79.01	77.36	77.37	78.23	78.19	**79.46**
	ReaL	84.61	84.47	84.62	84.56	85.01	83.39	83.36	84.28	84.17	**85.25**
	V2	66.73	66.39	66.02	66.35	66.82	65.17	64.52	66.13	65.96	**67.68**
ViT-B/16	ImageNet	80.12	80.26	80.32	80.29	79.85	78.95	78.50	79.54	79.90	**80.77**
	ReaL	85.46	85.45	85.44	85.48	85.16	84.76	84.49	85.10	85.36	**86.15**
	V2	68.14	67.76	68.46	68.19	68.48	67.46	65.95	68.07	68.20	**69.19**

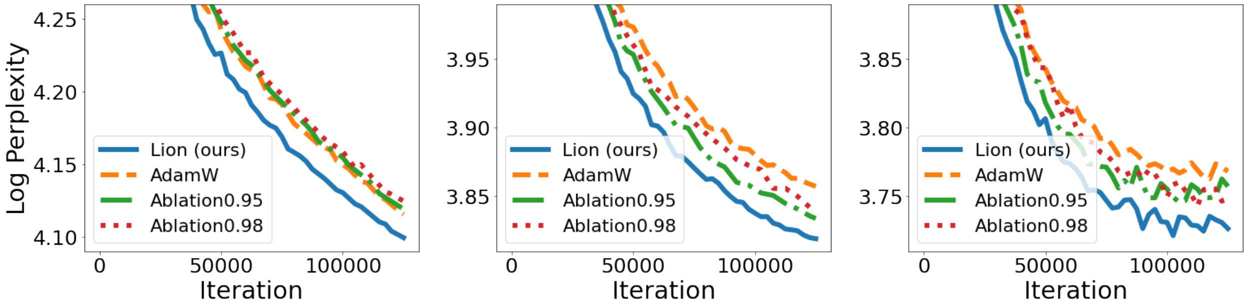
thoroughly tune the peak learning rate lr and decoupled weight decay λ (Loshchilov and Hutter, 2019) of every optimizer, while other hyperparameters are set as the default values in Optax.[1] As shown in Table 6.8, Lion is still the best performing one. We notice that there is no clear winner amongst the baselines. AMSGrad performs the best on ViT-S/16 but the worst on ViT-B/16. The inferior performance of PowerSign and AddSign compared to other optimizers is consistent with previous observations that automatically discovered optimizers have difficulty generalizing to real-world learning tasks. Figure 6.10 further shows that the learning curves of the five adaptive optimizers are pretty similar, whereas Lion has a unique one that learns faster.

6.3.6 Ablations

Momentum tracking To ablate the effects of both β_1 and β_2, we compare to a simple update rule: `m = interp(g, m, β); update = sign(m)`. Two optimizers, Ablation$_{0.9}$ and Ablation$_{0.99}$, are created with β values of 0.9 and 0.99 respectively. Illustrated by Table 6.8, the two ablated optimization algorithms perform worse than all five compared baselines,

[1] https://github.com/deepmind/optax

Figure 6.11: Log perplexity of the small (**Left**), medium (**Middle**), and large (**Right**) size Transformer on PG-19. Since $\beta_1 = 0.95, \beta_2 = 0.98$ in Lion when performing language modeling, we compare to Ablation$_{0.95}$ and Ablation$_{0.98}$ with $\beta = 0.95$ and $\beta = 0.98$, respectively (see Section 6.3.6 for the definition). Lion is still the best-performing one.



let alone our Lion. Further ablation studies on the language modeling task (as depicted in Figure 6.11) yield similar conclusions. Those results validate the effectiveness and necessity of using two linear interpolation functions, letting Lion to remember longer gradient history meanwhile assign a higher weight to the current gradient.

Effect of batch size Some may question whether Lion requires a large batch size to accurately determine the direction due to the added noise from the sign operation. To address this concern, we train a ViT-B/16 model on ImageNet using various batch sizes while maintaining the total training epoch as 300, and incorporating RandAug and Mixup techniques. As shown in Figure 6.12 (Left), the optimal batch size for AdamW is 256, while for Lion is 4,096. This indicates that Lion indeed prefers a larger batch size, but its performance remains robust even with a small 64 batch size. Furthermore, when the batch size enlarges to 32K, leading to only 11K training steps, Lion achieves a significant 2.5% accuracy gain over AdamW (77.9% vs. 75.4%), demonstrating its effectiveness in the large batch training setting.

Figure 6.12: **Left**: Ablation for the effect of batch size. Lion prefers a larger batch than AdamW. ImageNet accuracy of ViT-B/16 trained from scratch when we vary lr and λ for AdamW (**Middle**) and Lion (**Right**). Lion is more robust to different hyperparameter choices.

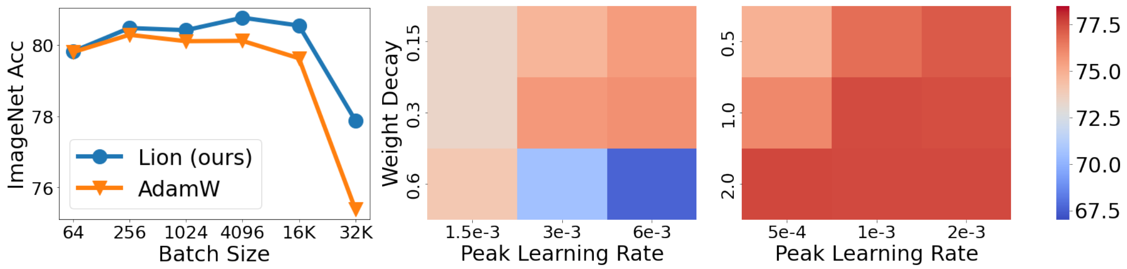


Table 6.9: Architecture details for language modeling.

Model	#Params	n_{layers}	d_{model}	n_{heads}	d_{head}
		Small-scale			
Small	110M	12	768	12	64
Medium	336M	24	1024	16	64
Large	731M	24	1536	16	96
		Large-scale			
1.1B	1.07B	24	1536	16	96
2.1B	2.14B	32	2048	16	128
7.5B	7.49B	32	4096	32	128

Table 6.10: Training error L_{train} and landscape flatness $L^{\mathcal{N}}_{train}$ of ViT-B/16 trained from scratch on ImageNet.

Optimizer	AdamW	Lion
ImageNet	75.48	77.44
ReaL	80.64	82.57
V2	61.87	64.81
L_{train}	0.61	0.75
$L^{\mathcal{N}}_{train}$	3.74	1.37

Analysis of Loss Landscape In this section, we try to understand why our Lion optimizer achieves better generalization than AdamW from the lens of loss geometry. The convergence to a smooth landscape has been shown to benefit the generalization of deep neural networks (Chen and Hsieh, 2020; Chen et al., 2022; Foret et al., 2021a; Keskar et al., 2017). Following Chen et al. (2022), we measure the landscape flatness at convergence by $L^{\mathcal{N}}_{train} = \mathbb{E}_{\epsilon\sim\mathcal{N}}[L_{train}(w+\epsilon)]$ (average over 1K random noises) in Table 6.10. We observe that the ViT-B/16 trained by AdamW enjoys a smaller training error L_{train}. However, Lion can enable ViT to converge to flatter regions, as it helps the model retain comparably lower error against Gaussian perturbations.

6.4 Hyperparameter Tuning

To ensure a fair comparison, we tune the peak learning rate lr and decoupled weight decay λ for both AdamW (Adafactor) and our Lion using a logarithmic scale. The default values for β_1 and β_2 in AdamW are set as 0.9 and 0.999, respectively, with an ϵ of $1e-8$, while in Lion, the default values for β_1 and β_2 are discovered through the program search process and set as 0.9 and 0.99, respectively. We only tune those hyperparameters in Section 6.3.4, where $\beta_1 = 0.9$, $\beta_2 = 0.99$ in AdamW, and $\beta_1 = 0.95$, $\beta_2 = 0.98$ in Lion. In our experience, reducing β_2 results in shorter memorization of historical information and *enhanced training stability*. Additionally, the ϵ in AdamW is set as $1e-6$ instead of the default $1e-8$ as it improves stability in our experiments, similar to the observations in RoBERTa (Liu et al., 2019).

The update generated by Lion is an element-wise binary ± 1, as a result of the sign operation, therefore it has a larger norm than those generated by other optimizers. Based on our experience, *a suitable learning rate for Lion is typically 3-10x smaller than that for AdamW.* Note that the initial value, peak value, and end value of the learning rate should be changed *simultaneously* with the same ratio compared to AdamW. We *do not* modify other training settings such as the learning rate schedule, gradient and update clipping. Since the effective weight decay is `lr * λ`: `update += w * λ`; `update *= lr`, *the value of λ used for Lion is 3-10x larger than that for AdamW in order to maintain a similar strength.* For instance,

- $lr = 1e-4$, $\lambda = 10.0$ in Lion and $lr = 1e-3$, $\lambda = 1.0$ in AdamW when training ViT-B/16 on ImageNet with strong augmentations,
- $lr = 3e-5$, $\lambda = 0.1$ in Lion and $lr = 3e-4$, $\lambda = 0.01$ in AdamW for diffusion models,
- $lr = 1e-4$, $\lambda = 0.01$ in Lion and $lr = 1e-3$, $\lambda = 0.001$ in Adafactor for the 7.5B language modeling.

Apart from the peak performance, the sensitivity to hyperparameters and the difficulty in tuning them are also critical for the adoption of an optimizer in practice. In Figure 6.12 (Middle and Right), we alter both lr and λ when training ViT-B/16 from scratch on ImageNet. Suggested by the heatmaps, Lion is more robust to different hyperparameter choices compared to AdamW.

6.5 Limitations

Limitations of search Despite the efforts to make the search space less restrictive, it remains inspired by the popular first-order optimization algorithms, leading to a bias towards similar algorithms. It also lacks the functions required to construct advanced second-order algorithms (Anil et al., 2020; Gupta et al., 2018; Martens and Grosse, 2015). The search cost is still quite large and the algorithm simplification requires manual intervention. Further reducing the bias in the search space to discover more novel algorithms and improving the search efficiency are important future directions. The current program structure is quite simplistic, as we do not find a good usage of more advanced program constructs such as conditional, loop statements, and defining new functions. Exploring how to incorporate these elements has the potential to unlock new possibilities.

Limitations of Lion While we endeavour to evaluate Lion on as many tasks as possible, the assessment is limited to the chosen tasks. On vision tasks, the discrepancies between Lion, AdamW, and momentum SGD are pretty small on ResNets, likely due to the fact that ConvNets are easier to optimize compared to Transformers. The performance gain brought by Lion decreases when strong augmentations are utilized. There are also several tasks where Lion performs similarly to AdamW, including: (1) the Imagen text-to-image base model, (2) the perplexity of autoregressive language model trained on the large-scale internal dataset, which is arguably a more reliable metric the in-context learning benchmarks, and (3) masked language modeling on C4. These tasks have a common characteristic in that the datasets are massive and of high quality, which results in a reduced difference between optimizers. Another potential limitation is the batch size. Though people often scale up the batch size to enable more parallelism, it is likely that Lion performs no better than AdamW if the batch size is small (<64). Additional, Lion still requires momentum tracking in `bfloat16`, which can be expensive for training giant models. One potential solution is to factorize the momentum to save memory.

CHAPTER 7

Conclusions

This thesis has presented a comprehensive study on two critical domains within the broader landscape of Automated Machine Learning (AutoML): Neural Architecture Search (NAS) and the automated discovery of optimization algorithms. We have showcased that both of these aspects are crucial in realizing the complete potential of AutoML and for its broader adoption in academia and industry.

In the first part of the thesis, we focused on differentiable NAS methods, specifically targeting the stability and robustness issues associated with the DARTS framework. Our novel approaches, built upon perturbation-based regularization and architecture distribution learning, offer a more reliable and robust framework for architecture search. We demonstrated the practical utility of these advancements in various applications, such as recommender systems and knowledge graphs. By enhancing the dependability of differentiable NAS methods, we pave the way for broader applications of AutoML, especially in domains where stability and reliability are of paramount importance.

The latter part of the thesis explored a paradigm shift within machine learning research—the automated discovery of optimization algorithms. This shift is reflective of a larger change in research focus, moving from developing entirely new architectures to refining and optimizing existing ones. We have argued that the discovery of new optimization algorithms holds as much importance as architecture search, if not more, especially in the era of Transformer architectures. Our novel method, Lion, is an evolved optimization algorithm that offers a compelling new direction for the field. It not only simplifies the optimization process but also demonstrates effectiveness across diverse neural architectures, underscoring its potential for broad applicability.

The advances presented in this thesis are steps towards fulfilling the promise of AutoML: democratizing machine learning and significantly reducing the manual effort and expertise required in the deployment of robust, high-performing models. Through these contributions, we aim to accelerate the pace of innovation and enable faster, more efficient solutions to the complex challenges that machine learning aims to address. Yet, several avenues remain open for future exploration. In the realm of NAS, further research is needed to expand our understanding of stability in architecture search methods, potentially incorporating real-world constraints such as latency and energy efficiency. For optimization algorithms, extending Lion to cater to a broader range of machine learning tasks will be an interesting pursuit. Moreover, a hybrid approach that combines both architecture search and optimization algorithm discovery could offer a holistic solution for constructing high-performing machine learning pipelines.

Bibliography

C. Aggarwal. *Recommender systems: the textbook*. Springer, 2017.

Charles Sutton Akash Srivastava. Autoencoding variational inference for topic models. In *International Conference on Learning Representations*, 2017. URL `https://arxiv.org/abs/1703.01488`.

Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *arXiv preprint arXiv:2104.11178*, 2021.

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning, 2020.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. *arXiv preprint arXiv:2103.15691*, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the*

35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 404–413. PMLR, 10–15 Jul 2018.

Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 459–468. JMLR.org, 2017.

Irwan Bello, William Fedus, Xianzhi Du, Ekin D. Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies, 2021.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/bender18a.html`.

Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8): 1889–1900, 2000.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13 (Feb):281–305, 2012.

J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8 (1):014008, 2015.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 560–569. PMLR, 10–15 Jul 2018.

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020a.

Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet?, 2020b.

Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer New York, 2016.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020. doi: 10.1609/aaai.v34i05.6239.

M. Blondel, A. Fujino, N. Ueda, and M. Ishihata. Higher-order factorization machines. In *NIPS*, pages 3351–3359, 2016.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th*

International Conference on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, pages 557–565. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/botev17a.html.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Markus Brameier, Wolfgang Banzhaf, and Wolfgang Banzhaf. *Linear genetic programming*, volume 1. Springer, 2007.

Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rydeCEhs-.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Rebekka Burkholz and Alina Dubatovka. Initialization of relus for dynamical isometry. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/d9731321ef4e063ebbee79298fa36f56-Paper.pdf.

Yoshua Bengio Caglar Gulcehre, Sarath Chandar. Memory augmented neural networks with wormhole connections, 2017.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018a.

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 678–687, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018b. PMLR. URL `http://proceedings.mlr.press/v80/cai18a.html`.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HylVB3AqYm`.

E. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021.

Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search, 2019.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations*, 2017.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *ArXiv*, abs/2103.12828, 2021a.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016. URL `http://arxiv.org/abs/1511.05641`.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. URL `http://proceedings.mlr.press/v119/chen20j.html`.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021b. URL `https://openreview.net/forum?id=Cnon5ezMHtu`.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1554–1565. PMLR, 13–18 Jul 2020.

Xiangning Chen, Cihang Xie, Mingxing Tan, Li Zhang, Cho-Jui Hsieh, and Boqing Gong. Robust and accurate object detection via adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16622–16631, June 2021c.

Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. In *International Conference on Learning Representations*, 2022.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303, 2019.

Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers, 2021d.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, and M. Ispir. Wide & deep learning for recommender systems. Technical report, Recsys Workshop, 2016.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets, 2017.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search, 2019.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3008–3017, 2020. doi: 10.1109/CVPRW50498.2020.00359.

M. F. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *RecSys*, pages 101–109. ACM, 2019.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3965–3977. Curran Associates, Inc., 2021.

Jon D. McAuliffe David M. Blei, Alp Kucukelbir. Variational inference: A review for statisticians, 2016.

Michael I. Jordan David M. Blei, Andrew Y. Ng. Latent dirichlet allocation. *The Journal of Machine Learning Research*, Mar 2003. ISSN 1532-4435. doi: 10.1162/jmlr.2003.3.4-5.993. URL `http://dx.doi.org/10.1109/CVPR.2017.243`.

Marie-Catherine de Marneffe, Mandy Simons, and Judith Tonhauser. The commitmentbank: Investigating projection in naturally occurring discourse. *Proceedings of Sinn und Bedeutung*, 23(2):107–124, Jul. 2019. doi: 10.18148/sub/2019.v23i2.601.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the*

2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.

Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021a. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021b.

Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *Proceedings of the 4th International Conference on Learning Representations*, pages 1–4, 2016.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 17–23 Jul 2022.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2010.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In Gal Elidan, Kristian Kersting, and Alexander T. Ihler, editors, *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press, 2017. URL `http://auai.org/uai2017/proceedings/papers/173.pdf`.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ByME42AqK7`.

Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *NIPS*, pages 2962–2970, 2015.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021a.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021b. URL `https://openreview.net/forum?id=6Tm1mposlrM`.

Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004.

R. Gemulla, E. Nijkamp, P. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, pages 69–77, 2011.

Ryan Gillard, Stephen Jonany, Yingjie Miao, Michael Munn, Connal de Souza, Jonathan Dungay, Chen Liang, David R. So, Quoc V. Le, and Esteban Real. Unified functional hashing in automatic machine learning, 2023.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *FOGA*, 1991.

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

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. SemEval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1:*

Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012), pages 394–398, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. Wiki-40B: Multilingual language model dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2440–2452, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1842–1850. PMLR, 10–15 Jul 2018.

Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.668. URL `http://dx.doi.org/10.1109/CVPR.2017.668`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. *WWW*, 2017.

X. He, X. Du, X. Wang, F. Tian, J. Tang, and T.-S. Chua. Outer product-based neural collaborative filtering. In *IJCAI*, pages 2227–2233, 2018.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJz6tiCqYm`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization, 2020.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8340–8349, October 2021a.

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15262–15271, June 2021b.

J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR*, pages 230–237, 1999.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

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

C.-K. Hsieh, L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin. Collaborative metric learning. In *WWW*, pages 193–201. International World Wide Web Conferences Steering Committee, 2017.

Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining, 2020.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9099–9117. PMLR, 17–23 Jul 2022.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.243. URL `http://dx.doi.org/10.1109/CVPR.2017.243`.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amost Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SkgEaj05t7`.

H. Ji, C. Liu, Z. Shen, and Y. Xu. Robust video denoising using low rank matrix completion. In *CVPR*, pages 1791–1798, 2010.

Weonyoung Joo, Wonsung Lee, Sungrae Park, , and Il-Chul Moon. Dirichlet variational autoencoder, 2019. URL `https://openreview.net/forum?id=rkgsvoA9K7`.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147.

A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Recsys*, pages 79–86, 2010.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=H1oyRlYgg`.

Samuel Kessler, Vu Nguyen, Stefan Zohren, and Stephen Roberts. Hierarchical indian buffet neural networks for bayesian continual learning, 2019.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1023.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18661–18673. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/d89a66c7c80a29b1bdbab0f2a1a94af8-Paper.pdf`.

D. Kim, C. Park, J. Oh, S. Lee, and H. Yu. Convolutional matrix factorization for document context-aware recommendation. In *RecSys*, pages 233–240, 2016.

M. Kim and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *SDM*, pages 47–58. SIAM, 2011.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *ICLR*, 2017.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine*

Learning Research, pages 2698–2707. PMLR, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/kleinberg18a.html`.

T.G. Kolda and B. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3): 455–500, 2009.

Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 2008.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4:87–112, 1994.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7: 452–466, 2019.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1082.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning," in international conference on learning representations. In *ICLR*, 2020.

T. Lei, X. Wang, and H. Liu. Uncoverning groups via heterogeneous interaction analysis. In *ICDM*, pages 503–512, 2009.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press, 2012. ISBN 9781577355601.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf`.

Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations*, 2017.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search, 2020.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping, 2019.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *Lecture Notes in Computer Science*, page 19–35, 2018a. ISSN 1611-3349. doi: 10.1007/978-3-030-01246-5_2. URL `http://dx.doi.org/10.1007/978-3-030-01246-5_2`.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *ICLR*, 2018b.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search, 2017.

Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021a.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. In *NeurIPS*, 2018a.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7816–7827. Curran Associates, Inc., 2018b. URL `http://papers.nips.cc/paper/8007-neural-architecture-optimization.pdf`.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *The European Conference on Computer Vision (ECCV)*, September 2018.

David J. C. MacKay. Choice of basis for laplace approximation. *Machine Language*, October 1998. doi: 10.1023/A:1007558615313. URL `https://link.springer.com/article/10.1023/A:1007558615313`.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018a.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018b. URL `https://openreview.net/forum?id=rJzIBfZAb`.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

Fritz Obermeyer Martin Jankowiak. Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, 2018. URL `https://arxiv.org/abs/1806.01851`.

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BylQSxHFwr`.

Luke Melas-Kyriazi. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet. *arXiv preprint arXiv:2105.02723*, 2021.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4556–4565. PMLR, 09–15 Jun 2019.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. Velo: Training versatile learned optimizers by scaling up, 2022.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and et al. Evolving deep neural networks. *Artificial Intelligence in the Age of Neural Networks and Brain*

Computing, page 293–312, 2019. doi: 10.1016/b978-0-12-815480-9.00015-3. URL `http://dx.doi.org/10.1016/B978-0-12-815480-9.00015-3`.

A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2008.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1098.

Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Lukasz Kaiser, Karol Kurach, Ilya Sutskever, and James Martens. Adding gradient noise improves learning for very deep networks, 2017.

Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.441.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008. doi: 10.1109/ICVGIP.2008.47.

N. Parikh and S.P. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1 (3):123–231, 2013.

O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012a.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012b. doi: 10.1109/CVPR.2012.6248092.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/pham18a.html`.

Hieu Pham, Zihang Dai, Golnaz Ghiasi, Kenji Kawaguchi, Hanxiao Liu, Adams Wei Yu, Jiahui Yu, Yi-Ting Chen, Minh-Thang Luong, Yonghui Wu, Mingxing Tan, and Quoc V. Le. Combined scaling for open-vocabulary image classification, 2021.

Mohammad Taher Pilehvar and Jose Camacho-Collados. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1128.

Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2641–2649, 2015. doi: 10.1109/ICCV.2015.303.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

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

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2902–2911. JMLR.org, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4780–4789, Jul 2019a. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33014780. URL `http://dx.doi.org/10.1609/aaai.v33i01.33014780`.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, Jul. 2019b. doi: 10.1609/aaai.v33i01.33014780.

Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

B. Recht, M. Fazel, and P. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019a.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400. PMLR, 09–15 Jun 2019b.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

S. Rendle. Factorization machines with LibFM. *TIST*, 3(3):57, 2012.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993. doi: 10.1109/ICNN.1993.298623.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo-Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In Alice H. Oh, Alekh

Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr. 2020. doi: 10.1609/aaai.v34i05.6399.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf`.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR, 10–15 Jul 2018.

Yeonjong Shin and George Em Karniadakis. Trainability of relu networks and data-dependent initialization. *Journal of Machine Learning for Modeling and Computing*, 1(1):39–74, 2020. ISSN 2689-3967.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJxH22EKPS`.

Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJij4yg0Z`.

David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811. URL `https://doi.org/10.1162/106365602320169811`.

X. Su and T. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, 2009.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017a.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 843–852, 2017b. doi: 10.1109/ICCV.2017.97.

R. Sutton and A. Barto. *Reinforcement learning: An introduction.* MIT press, 1998.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL `http://arxiv.org/abs/1409.4842`.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training, 2021a.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention, 2021b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019a.

C. Wang and D. M Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456. ACM, 2011.

Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=PKubaeJkw3`.

Ruochen Wang, Yuanhao Xiong, Minhao Cheng, and Cho-Jui Hsieh. Efficient non-parametric optimizer search for diverse tasks. *arXiv preprint arXiv:2209.13575*, 2022.

Z. Wang, M.-J. Lai, Z. Lu, W. Fan, H. Davulcu, and J. Ye. Orthogonal rank-one matrix pursuit for low rank matrix completion. *SIAM JSC*, 37(1):A488–A514, 2015.

Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJx040EFvH`.

Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2958–2969. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1ef91c212e30e14bf125e9374262401f-Paper.pdf`.

Lechao Xiao, Jeffrey Pennington, and Samuel Schoenholz. Disentangling trainability and generalization in deep neural networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of

Proceedings of Machine Learning Research, pages 10462–10472. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/xiao20b.html.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L. Yuille, and Quoc V. Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2018.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylqooRqK7.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJlS634tPr.

H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, 2017.

Shiyang Yan, Jeremy S. Smith, Wenjin Lu, and Bailing Zhang. Hierarchical multi-scale attention networks for action recognition, 2017.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygrdpVKvr.

Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30.

Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/81c650caac28cdefce4de5ddc18befa0-Paper.pdf`.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=HkwZSG-CZ`.

Q. Yao and J. Kwok. Accelerated and inexact soft-impute for large-scale matrix and tensor completion. *TKDE*, 2018.

Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu. Differentiable neural architecture search via proximal iterations. Technical report, arXiv preprint arXiv:1905.13577, 2019.

Quanming Yao, Xiangning Chen, James T. Kwok, Yong Li, and Cho-Jui Hsieh. Efficient neural interaction function search for collaborative filtering. In *Proceedings of The Web Conference 2020*, WWW '20, page 1660–1670, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380237. URL `https://doi.org/10.1145/3366423.3380237`.

Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. Efficient neural architecture search via proximal iterations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (04):6664–6671, Apr. 2020b. doi: 10.1609/aaai.v34i04.6143. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6143`.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *ICML*, 2019.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Syx4wnEtvH`.

Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. Coca: Contrastive captioners are image-text foundation models. *Transactions on Machine Learning Research*, 2022.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1loF2NFwr`.

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet, 2021.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6022–6031, 2019. doi: 10.1109/ICCV.2019.00612.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=H1gDNyrKDS`.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020b. URL `https://openreview.net/forum?id=H1gDNyrKDS`.

Arber Zela, Julien Siems, and Frank Hutter. NAS-BENCH-1SHOT1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020c. URL `https://openreview.net/forum?id=SJx9ngStPH`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers, 2021.

Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. Lit: Zero-shot transfer with locked-image text tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18123–18133, June 2022.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018a. URL `https://openreview.net/forum?id=r1Ddp1-Rb`.

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J. Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? In *NeurIPS*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/b05b57f6add810d3b7490866d74c0053-Abstract.html`.

M. Zhang, H. Li, S. Pan, X. Chang, and S. Su. Overcoming multi-model forgetting in one-shot nas with diversity maximization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7806–7815, 2020.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. Record: Bridging the gap between human and machine commonsense reading comprehension, 2018b.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018c.

Y. Zhang, Q. Yao, W. Dai, and L. Chen. AutoKGE: Searching scoring functions for knowledge graph embedding. Technical report, arXiv preprint arXiv:1902.07638, 2019.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, 2018.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, pages 7603–7613, 2019. URL `http://proceedings.mlr.press/v97/zhou19e.html`.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18795–18806. Curran Associates, Inc., 2020.

B. Zoph, V. Vasudevan, J. Shlens, and Q. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2017.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00907. URL `http://dx.doi.org/10.1109/CVPR.2018.00907`.