UNIVERSITY OF CALIFORNIA,
IRVINE

Relaxation and Optimization for Automated Learning of Neural Network Architectures

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mathematics

by

Fanghui Xue

Dissertation Committee:
Professor Jack Xin, Chair
Professor Long Chen
Professor Knut Solna

2022

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Jack Xin, for his patience, encouragement and continuous support during my Ph.D. study. I have benefited a lot from his invaluable insights and inspirations throughout my research.

I would like to thank Professor Long Chen and Professor Knut Solna for joining my defense committee, and for their meaningful discussions and suggestions. I would also like to thank Professor Pierre Baldi, Professor Yifeng Yu and Professor Lizhi Sun for their helpful comments during my Ph.D. advancement.

I would like to thank Dr. Yingyong Qi, Dr. Shuai Zhang, Dr. Jiancheng Lyu and Biao Yang for the guidance and collaborations on many fascinating research projects. I would also like to thank Ziang Long, Kevin Bui, Zhijian Li, Yunling Zheng and all the other members in Professor Xin's lab.

Finally, I would like to express my appreciation to the Department of Mathematics, as well as all my friends, for their help and support in the last few years.

This work was partially supported by NSF grants DMS-1854434, DMS-1952644.

VITA

Fanghui Xue

EDUCATION

Doctor of Philosophy in Mathematics **2022**
University of California, Irvine *Irvine, CA*

Master of Science in Mathematical Risk Management **2018**
Georgia State University *Atlanta, GA*

Bachelor of Science in Mathematics **2016**
Fudan University *Shanghai, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2022**
University of California, Irvine *Irvine, CA*

TEACHING EXPERIENCE

Teaching Assistant **2018–2021**
University of California, Irvine *Irvine, CA*

ABSTRACT OF THE DISSERTATION

Relaxation and Optimization for Automated Learning of Neural Network Architectures

By

Fanghui Xue

Doctor of Philosophy in Mathematics

University of California, Irvine, 2022

Professor Jack Xin, Chair

Differentiable architecture search (DARTS) is an effective method for data-driven neural network design based on bilevel optimization. Despite its success in many architecture search tasks, there are still some concerns about the accuracy of the first-order DARTS and the efficiency of the second-order DARTS. In this article, we formulate a single level alternative and a relaxed architecture search (RARTS) method that utilizes the whole dataset in architecture learning via both data and network splitting, without involving mixed second derivatives of the corresponding loss functions like DARTS. The advantage of RARTS over DARTS is justified by a convergence theorem and an analytically solvable model. Moreover, RARTS outperforms DARTS and its variants in accuracy and search efficiency, as shown in adequate experiments on CIFAR-10 and ImageNet image classification datasets, and public architecture search benchmark like NATS-Bench.

Since network pruning is closely related to architecture search in the form of width and depth search, we have also adapted RARTS to width search and summarized it as a general framework. Experiments show that our method beats the previous benchmarks in PreResNet-164 pruning on CIFAR datasets. Additionally, it has been shown by many researchers that transformers perform as well as convolutional neural networks in many computer vision tasks. Meanwhile, the large computational costs of its attention module hinder further

studies and applications on edge devices. Some pruning methods have been developed to construct efficient vision transformers, but most of them have considered image classification tasks only. Inspired by these results, we extend our method for pruning vision transformer backbones on more complicated vision tasks like object detection, based on the search of transformer dimensions. Experiments on CIFAR-100 and COCO datasets show that the backbones with 20% or 40% dimensions/parameters pruned can have similar or even better performance than the unpruned models. Finally, we have also provided the complexity analysis and comparisons with the previous pruning methods.

Chapter 1

Introduction

Deep neural networks (DNNs) have been proved to be powerful and efficient in many fields by a large number of studies. Whereas the early DNNs have only a few layers [33], modern DNN architectures are getting more and more complicated. They may contain either thousands of layers, or parallel paths [49, 26], which makes it more difficult to design a network for specific problems manually. Researchers have then developed a technique called Neural Architecture Search (NAS) to automate the learning of neural network architectures, and have achieved state-of-the-art results on multiple tasks [69, 70]. Despite its high accuracy, searching for the optimal architecture may cost a huge overhead of thousands of GPU hours. Among the later studies, a Differentiable Architecture Search (DARTS) method has reduced the search cost greatly, while generating architectures of high performance at the same time. On the other hand, it has also been pointed out that DARTS may bring about a few problems like convergence issues and architecture collapse (i.e., generating a trivial architecture) [11, 24]. In particular, the full version of DARTS requires the computation of second-order derivatives of the loss function, consuming much more computational resources than its first-order counterpart. Motivated by the differentiable method and these unsolved problems, we will propose a first-order alternative which takes much less time to search for

the architecture.

Another topic which is closely related to architecture search is the pruning of DNNs, e.g., pruning the number of channels/filters/layers of convolutional neural networks (CNNs). Whereas CNNs have been widely used in many computer vision tasks such as image classification [32, 48, 26], semantic segmentation [8, 54] and object detection [34, 25], their high computational overheads are still unaffordable on many low-end edge devices. To solve this problem, researchers have manually constructed some efficient networks [49, 46]. Besides these manual designs, a number of approaches have also been proposed to generate light networks by pruning heavy ones [55, 38]. Since the width and the depth (i.e., the number of channels and layers) of a network can also be viewed as hyperparameters of its architecture design, network pruning problems are then formulated and solved in the way of architecture search [50, 18]. We are aware of these developments and would like to apply our first-order differentiable search method to solve the pruning problems.

Apart from CNNs, we have also witnessed a rapid growth of transformers [53] being used in many computer vision tasks like classification [20], segmentation [39], and object detection [7, 68]. Although they can achieve similar or better performance than CNNs, transformers usually cost more computational resources. So it is natural to imitate the pruning methods for CNNs to construct efficient transformers [67]. However, most of these pruning methods consider the image classification tasks only, whereas many transformers serve as the backbones for various vision tasks, going beyond the simple classification task. Inspired by the width search methods we have just mentioned, we would like to formulate the transformer pruning problem in a way of dimension search, extending it to a different vision task like object detection.

Here we list some useful notations and functions here, as they are going to be used in multiple

chapters. The ℓ_p norm of a vector $\boldsymbol{w} = (w_1, ..., w_d) \in \mathbb{R}^d$ is defined for $p \geq 1$:

$$\|\boldsymbol{w}\|_p = (\sum_i |w_i|^p)^{\frac{1}{p}}.$$

In this article, we can also define the norm for matrix or a tensor in the same way, i.e., taking $\frac{1}{p}$-th power of the sum of the p-th power of all the elements. For the case $p = 0$, we define:

$$\|\boldsymbol{w}\|_0 = \sum_i \mathbf{1}_{\mathbb{R}^\times}(w_i),$$

where $\mathbf{1}$ is the indicator function, and $\mathbb{R}^\times$ is the set of non-zero real numbers. In other words, ℓ_0 counts the number of non-zero elements. Usually, we use $\|\cdot\|$ to denote $\|\cdot\|_2$, unless pointed out specifically.

When training classification models, we usually compute the loss through the cross-entropy function [23], which is defined to be:

$$H(p, q) = -\sum_i p_i \log q_i, \tag{1.1}$$

where $p = (p_1, ..., p_C)$, $q = (q_1, ..., q_C)$, C is the number of classes, p_i and q_i represent the probabilities of the real and predicted labels belonging to the i-th class. Usually the probability p for the real label is one-hot, i.e., $p_i = \delta^{ij}$ if j is the correct class, with δ^{ij} the Kronecker delta function. Hence, the cross-entropy function can also be written as:

$$H_o(p, q) = -\log q_j, \tag{1.2}$$

where q_j is the probability of the predicted label belonging to the j-th class. Set $H(p) = -\sum_i p_i \log p_i$. The Kullback-Leibler divergence [23] (KL divergence) is defined to be:

$$D_{\mathrm{KL}}(p \,||\, q) = H(p, q) - H(p) = \sum_i p_i \log(p_i/q_i), \tag{1.3}$$

which is often used to show how the probability distribution p is different from q. For classification models where the probability of real label p is fixed, minimizing the KL divergence is equivalent to minimizing the cross-entropy.

Chapter 2

The Relaxed Architecture Search Method

In this chapter, we go over a class of relaxation methods for solving complicated optimization problems via variable splitting. Inspired by the previous works, we propose a relaxed method for automated learning of neural network architectures. In addition, we demonstrate in detail its corresponding gradient-based training algorithm and the convergence properties. We also consider the cases when there are regularization terms on model complexity. Comparisons of the proposed method with other architecture search methods are made through analytical examples and discussions.

2.1 Gradient Methods

2.1.1 Gradient Descent

Consider a differentiable function $\mathcal{L}(w)$ and the problem:

$$\min_{w} \mathcal{L}(w). \tag{2.1}$$

To learn the minimum, we can use iterative methods like *gradient descent* (see [23]):

$$w^{t+1} = w^t - \eta^t \nabla \mathcal{L}(w^t), \tag{2.2}$$

where w^t and η^t are the values of the variable and the *learning rate* at the t-th iteration. Suppose now $\mathcal{L}(w)$ is the loss function computed on a dataset of M training examples. That is to say,

$$\mathcal{L}(w) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i(w),$$

where $\mathcal{L}_i(w)$ is the loss function computed on the i-th example. Gradient descent can be computationally expensive for large M, since it needs to loop through all the examples in each iteration. *Mini-batch gradient descent* (see [23]) can reduce the computational cost by sampling uniformly each time a mini batch of fixed size m:

$$\mathcal{L}_t(w) = \frac{1}{m} \sum_{k=1}^{m} \mathcal{L}_{i_{t,k}}(w). \tag{2.3}$$

Here $\{i_{t,k}\}_{k=1}^{m}$ is the set of indices for the mini batch examples sampled in the t-th iteration. We update the variables using the following iteration:

$$w^{t+1} = w^t - \eta^t \nabla \mathcal{L}_t(w^t), \tag{2.4}$$

One shall be aware that the expectation of the gradient on the mini-batch loss function is equal to the gradient of the overall loss function:

$$\mathbb{E}[\nabla \mathcal{L}_t(w)] = \frac{1}{m} \sum_{k=1}^{m} \mathbb{E}[\nabla \mathcal{L}_{i_{t,k}}(w)] = \nabla \mathcal{L}(w). \tag{2.5}$$

Mini-batch gradient descent (especially when $m = 1$) is also called *stochastic gradient descent* (SGD) in many literature.

2.1.2 Relaxed Splitting Methods

We consider a regularized problem with a penalty function $\mathcal{P}(w)$:

$$\min_{w} \mathcal{L}(w) + \lambda \mathcal{P}(w),$$

where λ is a hyperparameter to control the scale of the penalty. If both $\mathcal{L}(w)$ and $\mathcal{P}(w)$ are smooth, we can learn w via gradient descent. However, there are cases where $\mathcal{L}(w)$ is smooth (or smooth almost everywhere) but $\mathcal{P}(w)$ is a non-smooth function like ℓ_0 norm [40]. Hence, a relaxed problem has been set up via variable spliting (RVSM) [16]:

$$\min_{w,u} \mathcal{L}(w) + \lambda \mathcal{P}(u) + \frac{\beta}{2} \|u - w\|_2^2,$$

where u has the same shape as w in terms of learnable parameters. Here β is a hyperparameter to control the distant between u and w. The parameters are learned in an alternating

way:

$$
\begin{aligned}
w^{t+1} &= w^t - \eta \nabla \mathcal{L}(w^t) - \eta\beta(w^t - u^t) \\
u^{t+1} &= \arg\min_u \{\lambda \mathcal{P}(u) + \frac{\beta}{2}\|u - w^t\|_2^2\}.
\end{aligned}
\tag{2.6}
$$

We note that the first step is simply gradient descent with respect to w. Although the second step could be more complicated, for $\mathcal{P}(u)$ being the ℓ_0 or ℓ_1 penalty, we have closed-form solutions [1, 13]:

$$
\begin{aligned}
\arg\min_u \{\lambda\|u\|_0 + \frac{\beta}{2}\|u - w\|_2^2\} &= H_{\lambda/\beta}(w) \\
\arg\min_u \{\lambda\|u\|_1 + \frac{\beta}{2}\|u - w\|_2^2\} &= S_{\lambda/\beta}(w),
\end{aligned}
$$

where H_γ and S_γ are defined componentwise:

$$
H_\gamma(w_i) = \begin{cases} 0 & \text{if } |w_i| \leq \sqrt{2\gamma} \\ w_i & \text{if } |w_i| > \sqrt{2\gamma}, \end{cases}
$$

and

$$
S_\gamma(w_i) = \begin{cases} w_i + \gamma & \text{if } w_i \leq -\gamma \\ 0 & \text{if } |\mathrm{w_i}| < \gamma \\ w_i - \gamma & \text{if } w_i \geq \gamma. \end{cases}
$$

Besides, there are many other functions eligible for the penalty in the second step of iteration (2.6), as long as it can be solved in closed-form. This includes an interpolation between ℓ_0 and ℓ_1 norms [64, 65], and the ℓ_1 norm (or weighted sum) on the scales of parameter groups [63, 59, 15, 4]. We will discuss their applications in the later chapters.

2.1.3 Gradient-based Architecture Search

Before we introduce the formulation of our relaxed method for architecture search of neural networks, we would like to briefly review the settings of Differentiable Architecture Search (DARTS) [37], which serves as a baseline gradient-based architecture search method. Suppose (w, α) is the pair of weight and architecture parameters taken from a model $f(X; w, \alpha)$ with input data X. We use (w, α) to represent the model, unless pointed out specifically. If the architecture α is fixed, we learn the weight parameters w of the model by minimizing a loss function $\mathcal{L}(w)$, which is basically what we do when training an ordinary neural network. DARTS has artfully relaxed the architecture parameters from binary to continuous, and learn the pair of two parameter groups (w, α) by considering the following bilevel problem [12]:

$$\begin{aligned} &\min_{\alpha} \mathcal{L}_2(w^*(\alpha), \alpha), \\ \text{where} \quad &w^*(\alpha) = \arg\min_{w} \mathcal{L}_1(w, \alpha). \end{aligned} \tag{2.7}$$

Here $\mathcal{L}_1$ and $\mathcal{L}_2$ are the two loss functions computed on $\mathcal{D}_1$ and $\mathcal{D}_2$, which are two non-overlapping half subsets of the dataset $\mathcal{D}$. Since the inner level problem $\min_w \mathcal{L}_1(w, \alpha)$ is hard to be fully solved with limited computational resources, we may approximate it by gradient descent in each step. DARTS has adopted data splitting because it is believed that joint training of both α and w via gradient descent on $\mathcal{D}$ by minimizing the overall loss function:

$$\mathcal{L}(w, \alpha) = \mathcal{L}_1(w, \alpha) + \mathcal{L}_2(w, \alpha)$$

can lead to overfitting [37, 24]. Therefore, the pair (w, α) in problem (2.7) is trained approximately in an alternating way:

- update weight w by descending along $\nabla_w \mathcal{L}_1(w, \alpha)$
- update architecture parameter α by descending along: $\nabla_\alpha \mathcal{L}_2(w - \xi \nabla_w \mathcal{L}_1(w, \alpha), \alpha)$

where $\xi \geq 0$ determines the complexity of approximation. If $\xi = 0$, we obtain first-order DARTS as the second step only involves the first-order derivatives of α. If $\xi > 0$, we obtain second-order DARTS as we need to further compute mixed derivatives $\nabla^2_{\alpha,w} \mathcal{L}_1(w, \alpha)$ which are produced by chain rule. It has been pointed out that second-order DARTS can have superposition effect [24], which means the approximation of the gradient of α is based on the approximation of the weight w one step ahead. This is believed to cause gradient errors and failures in finding optimal architectures. One shall also be aware that the direct computation of the second-order derivatives could be time-consuming, and hence may tend to use first-order DARTS. However, first-order DARTS updates the architecture parameters using half of the data only. We will present some evidences in the later sections to show that this can result in incorrect limits and worse performance. To overcome this issue, we consider approximating DARTS by introducing more relaxation and solve the relaxed problem via first-order gradient descent. Inspired by the data splitting of DARTS and the Relaxed Variable Splitting Methods in the previous subsection, we propose a Relaxed Architecture Search method (RARTS) using both data and network splitting.

2.2 Relaxed Architecture Search

2.2.1 The Vanilla Case

We consider the following relaxed Lagrangian without any further constraints:

$$\mathcal{L}(u, w, \alpha) := \mathcal{L}_p(u, \alpha) + \lambda \mathcal{L}_a(w, \alpha) + \frac{1}{2}\beta \|u - w\|_2^2, \tag{2.8}$$

where u and w are two groups of weight parameters for the primary model (u, α) and the auxiliary model (w, α). u and w have the same shape in terms of weight tensors, but they can have different initializations. The primary and the auxiliary models share the same architecture parameters, denoted by α. This splitting technique is called *network splitting*. The loss function $\mathcal{L}_p(u, \alpha)$ is computed from the model (u, α) fed with a proportion of data denoted by $\mathcal{D}_p$, while the loss function $\mathcal{L}_a(w, \alpha)$ is computed from the model (w, α) fed with a proportion of data denoted by $\mathcal{D}_a$. Here $\mathcal{D}_p$ and $\mathcal{D}_a$ can simply be two non-overlapping subsets of the original dataset $\mathcal{D}$, each of them taking half of the samples in $\mathcal{D}$. $\|u - w\|_2$ is the ℓ_2 norm, which penalizes the distance between the two groups of weight parameters. λ and β are two non-negative hyperparameters which control the scales of the loss function for the auxiliary model (w, α), and the ℓ_2 penalty.

We want to search an architecture α along with the weight parameters u and w, so that the relaxed Lagrangian $\mathcal{L}(u, w, \alpha)$ in equation (2.8) can be small. We develop an alternating gradient descent algorithm to solve the problem:

$$\min_{u,w,\alpha} \mathcal{L}(u, w, \alpha). \tag{2.9}$$

In each iteration, we update the three groups of parameters via gradient descent in the Gauss-Seidel [22] pattern, i.e., only one group of parameters is updated in each step:

$$\begin{aligned} w^{t+1} &= w^t - \eta_w^t \nabla_w \mathcal{L}(u^t, w^t, \alpha^t) \\ u^{t+1} &= u^t - \eta_u^t \nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t) \\ \alpha^{t+1} &= \alpha^t - \eta_\alpha^t \nabla_\alpha \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^t), \end{aligned} \tag{2.10}$$

where (u^t, w^t, α^t) and $(u^{t+1}, w^{t+1}, \alpha^{t+1})$ stand for the values of the three group of parameters at iterations t and $t + 1$. Gradients of $\mathcal{L}(u, w, \alpha)$ are computed with respect to w, u and α

in each of the three steps, adjusted by the learning rates η_w^t, η_u^t and η_α^t. The three learning rates are not necessarily identical. Typically, a learning rate schedule is set up so that the learning rates can be adjusted during the training of the model. We summarize the Relaxed Architecture Search algorithm for the vanilla case (without regularization) in Algorithm 1.

Algorithm 1: Relaxed Architecture Search (RARTS)

Input: the number of iterations N, the hyperparameters λ and β, a learning rate schedule $(\eta_w^t, \eta_u^t, \eta_\alpha^t)$, initialization of the weight parameters w^0, u^0 and the architecture parameters α^0.

Output: α^*, the architecture we want

Split the dataset $\mathcal{D}$ into two subsets $\mathcal{D}_p$ and $\mathcal{D}_a$.

for $t = 0, 1, ..., N$ **do**

Compute $\mathcal{L}_p$ and $\mathcal{L}_a$ on $\mathcal{D}_p$ and $\mathcal{D}_a$, respectively, and then compute $\mathcal{L}$ using equation (2.8)

Update the parameters via gradient descent:

$w^{t+1} = w^t - \eta_w^t \nabla_w \mathcal{L}(u^t, w^t, \alpha^t)$

$u^{t+1} = u^t - \eta_u^t \nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t)$

$\alpha^{t+1} = \alpha^t - \eta_\alpha^t \nabla_\alpha \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^t)$

end

2.2.2 Model Complexity and Regularization

Apart from the vanilla case when there are no further constraints to the problem (2.9), we also consider a problem constrained by model complexity. We introduce a penalty function $\mathcal{P}(\alpha)$ to represent model complexity and consider the following regularized Lagrangian:

$$\mathcal{L}(u, w, \alpha) := \mathcal{L}_p(u, \alpha) + \lambda \mathcal{L}_a(w, \alpha) + \frac{1}{2}\beta \|u - w\|_2^2 + \gamma \mathcal{P}(\alpha), \tag{2.11}$$

where $\gamma \geq 0$ is a hyperparameter to control the scale of the penalty. One shall be aware that the penalty function $\mathcal{P}(\alpha)$ depends purely on the architecture parameters α. Again, we use Algorithm 1 to solve the minimization problem (2.9) with few modifications. The only difference is that the Lagrangian $\mathcal{L}(u, w, \alpha)$ for the vanilla case is now replaced by the regularized Lagrangian (2.11). The penalty function $\mathcal{P}(\alpha)$ only impacts the third step of equation (2.10), since the other two steps do not compute gradients with respect to α. This is clear if we write out explicitly the gradient of each term in iteration (2.10):

$$\begin{aligned} w^{t+1} &= w^t - \eta_w^t \lambda \nabla_w \mathcal{L}_a(w^t, \alpha^t) - \eta_w^t \beta (w^t - u^t) \\ u^{t+1} &= u^t - \eta_u^t \nabla_u \mathcal{L}_p(u^t, \alpha^t) - \eta_u^t \beta (u^t - w^{t+1}) \\ \alpha^{t+1} &= \alpha^t - \eta_\alpha^t \lambda \nabla_\alpha \mathcal{L}_a(w^{t+1}, \alpha^t) - \eta_\alpha^t \nabla_\alpha \mathcal{L}_p(u^{t+1}, \alpha^t) - \eta_\alpha^t \gamma \nabla_\alpha \mathcal{P}(\alpha^t). \end{aligned} \tag{2.12}$$

Note that the Lagrangian (2.8) is a special case of (2.11) when $\gamma = 0$.

2.3 Main Results

We present the convergence results of Algorithm 1 in this section. First, we list some conditions which help to establish the statement of our convergence theorem.

(i) A differentiable function $f(x)$ defined on $\mathbb{R}^d$ satisfies *Lipschitz gradient property* if there is a number L so that for $x, y \in \mathbb{R}^d$

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|. \tag{2.13}$$

(ii) A function $f(x)$ defined on $\mathbb{R}^d$ is said to be *coercive* (see [44]) if

$$\lim_{\|x\| \to +\infty} f(x) = +\infty.$$

The following lemma from [41] can help estimate the descending of the loss function in many gradient descent algorithms. We include its proof for completeness.

Lemma 2.1. *Suppose a differentiable function $f(x)$ satisfies Lipschitz gradient property on $\mathbb{R}^d$. Then we have:*

$$f(x) - f(y) \leq \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2. \tag{2.14}$$

Proof. We set $g(t) = f(tx + (1-t)y)$. We have

$$\begin{aligned}
f(x) - f(y) &= g(1) - g(0) \\
&= \int_0^1 g'(t)\,\mathrm{d}t \\
&= \int_0^1 \langle \nabla f(tx + (1-t)y), x - y \rangle\,\mathrm{d}t \\
&= \int_0^1 \langle \nabla f(tx + (1-t)y) - \nabla f(y), x - y \rangle\,\mathrm{d}t + \langle \nabla f(y), x - y \rangle \\
&\leq \int_0^1 \|\nabla f(tx + (1-t)y) - \nabla f(y)\|\|x - y\|\mathrm{d}t + \langle \nabla f(y), x - y \rangle \\
&\leq \int_0^1 Lt\|x - y\|^2\mathrm{d}t + \langle \nabla f(y), x - y \rangle \\
&= \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2.
\end{aligned}$$

We have applied Cauchy-Schwarz inequality and Lipschitz gradient property (2.13) to obtain the first and second inequalities in the proof. □

2.3.1 Convergence and Equilibrium

We introduce the main convergence results of Algorithm 1.

Theorem 2.2. *Suppose there is a number L such that the loss functions $\mathcal{L}_p(u, \alpha)$ and $\mathcal{L}_a(w, \alpha)$ satisfy the Lipschitz gradient property (2.13), where the gradients are taken with*

respect to (u, α) *and* (w, α). *Suppose for the same* L, *the penalty function* $\mathcal{P}(\alpha)$ *also satisfies the Lipschitz gradient property (2.13), without loss of generality. If there is a number* T *such that for* $t \geq T$, *the learning rates satisfy:*

$$\begin{aligned} \eta_y^t &< \frac{1}{2}\left[\frac{\beta}{2} + L\right]^{-1} := c_1, \\ \eta_w^t &< \frac{1}{2}\left[\frac{\beta}{2} + \lambda L\right]^{-1} := c_2, \\ \eta_\alpha^t &< \frac{1}{2}\left[(1 + \lambda + \frac{\gamma}{2})L\right]^{-1} := c_3, \end{aligned} \tag{2.15}$$

and approach nonzero limits at large t, *the Lagrangian function* $\mathcal{L}(u, w, \alpha)$ *defined by equation (2.8) or (2.11) is descending on the iterations of (2.10), i.e.,* $\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) \leq \mathcal{L}(u^t, w^t, \alpha^t)$ *for* $t \geq T$.

Proof. As the Lagrangian in equation (2.8) is a special case of that in equation (2.11), we only need to prove the theorem for (2.11). Since

$$\mathcal{L}(u^t, w^t, \alpha^t) = \mathcal{L}_p(u^t, \alpha^t) + \lambda\, \mathcal{L}_a(w^t, \alpha^t) + \frac{1}{2}\beta\, \|u^t - w^t\|^2 + \gamma\, \mathcal{P}(\alpha^t),$$

together with the inequality from Lemma 2.1, we have

$$\begin{aligned} & \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t) \\ = \;& \mathcal{L}_p(u^{t+1}, \alpha^{t+1}) - \mathcal{L}_p(u^t, \alpha^t) + \lambda \left(\mathcal{L}_a(w^{t+1}, \alpha^{t+1}) - \mathcal{L}_a(w^t, \alpha^t)\right) \\ & + \frac{1}{2}\beta \left(\|u^{t+1} - w^{t+1}\|^2 - \|u^t - w^t\|^2\right) + \gamma \left(\mathcal{P}(\alpha^{t+1}) - \mathcal{P}(\alpha^t)\right) \\ \leq \;& \langle \nabla_{u,\alpha}\, \mathcal{L}_p(u^t, \alpha^t), (u^{t+1} - u^t, \alpha^{t+1} - \alpha^t)\rangle + \frac{L}{2}\|(u^{t+1} - u^t, \alpha^{t+1} - \alpha^t)\|^2 \\ & + \lambda \,\langle \nabla_{w,\alpha}\, \mathcal{L}_a(w^t, \alpha^t), (w^{t+1} - w^t, \alpha^{t+1} - \alpha^t)\rangle + \frac{\lambda L}{2}\|(w^{t+1} - w^t, \alpha^{t+1} - \alpha^t)\|^2 \\ & + \gamma \,\langle \nabla_\alpha\, \mathcal{P}(\alpha^t), \alpha^{t+1} - \alpha^t\rangle + \frac{\gamma L}{2}\|\alpha^{t+1} - \alpha^t\|^2 + \frac{\beta}{2}\big(\|u^{t+1} - w^{t+1}\|^2 - \|u^t - w^t\|^2\big). \end{aligned}$$

Substituting for the (w,u)-gradients from the iterations (2.12), we continue:

$$
\begin{aligned}
& \mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1}) - \mathcal{L}(u^t,w^t,\alpha^t) \\
\leq\ & -(\eta_u^t)^{-1}\,\langle u^{t+1}-u^t+\eta_u^t\,\beta\,(u^t-w^{t+1}),u^{t+1}-u^t\rangle \\
& +\lambda\,(-\lambda\eta_w^t)^{-1}\,\langle w^{t+1}-w^t+\eta_w^t\,\beta\,(w^t-u^t),w^{t+1}-w^t\rangle \\
& +\frac{L}{2}\|u^{t+1}-u^t\|^2+\frac{\lambda L}{2}\|w^{t+1}-w^t\|^2 \\
& +\frac{\beta}{2}\big(\|u^{t+1}-w^{t+1}\|^2-\|u^t-w^t\|^2\big)+\frac{L(1+\lambda+\gamma)}{2}\|\alpha^{t+1}-\alpha^t\|^2 \\
& +\langle\nabla_\alpha\,\mathcal{L}_p(u^t,\alpha^t)+\lambda\nabla_\alpha\,\mathcal{L}_a(w^t,\alpha^t)+\gamma\nabla_\alpha\,\mathcal{P}(\alpha^t),\alpha^{t+1}-\alpha^t\rangle \\
=\ & \big(-(\eta_u^t)^{-1}+L/2\big)\,\|u^{t+1}-u^t\|^2+\big(-(\eta_w^t)^{-1}+\lambda L/2\big)\,\|w^{t+1}-w^t\|^2 \\
& -\beta\big(\langle u^t-w^{t+1},u^{t+1}-u^t\rangle+\langle w^t-u^t,w^{t+1}-w^t\rangle\big) \\
& +\frac{\beta}{2}\big(\|u^{t+1}-w^{t+1}\|^2-\|u^t-w^t\|^2\big)+\frac{L(1+\lambda+\gamma)}{2}\|\alpha^{t+1}-\alpha^t\|^2 \\
& +\langle\nabla_\alpha\,\mathcal{L}_p(u^t,\alpha^t)+\lambda\nabla_\alpha\,\mathcal{L}_a(w^t,\alpha^t)+\gamma\nabla_\alpha\,\mathcal{P}(\alpha^t),\alpha^{t+1}-\alpha^t\rangle. \qquad (2.16)
\end{aligned}
$$

We note the following identity

$$
\begin{aligned}
& \|u^{t+1}-w^{t+1}\|^2 \\
=\ & \|u^{t+1}-w^t+w^t-w^{t+1}\|^2 \\
=\ & \|u^{t+1}-w^t\|^2+2\langle u^{t+1}-w^t,w^t-w^{t+1}\rangle+\|w^t-w^{t+1}\|^2,
\end{aligned}
$$

where

$$
\begin{aligned}
& \|u^{t+1}-w^t\|^2 \\
=\ & \|-w^t+u^t-u^t+u^{t+1}\|^2 \\
=\ & \|u^t-w^t\|^2+2\langle u^t-w^t,u^{t+1}-u^t\rangle+\|u^{t+1}-u^t\|^2.
\end{aligned}
$$

Upon substitution of the above in the right hand side of (2.16), we find that:

$$\begin{aligned}
& \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t) \\
\leq\ & \big(-(\eta_u^t)^{-1} + L/2 + \beta/2 \big) \|u^{t+1} - u^t\|^2 + \big(-(\eta_w^t)^{-1} + \lambda L/2 + \beta/2 \big) \|w^{t+1} - w^t\|^2 \\
& + \beta \langle w^{t+1} - w^t, u^{t+1} - u^t \rangle + \beta \langle u^{t+1} - u^t, w^t - w^{t+1} \rangle + \frac{L(1+\lambda+\gamma)}{2} \|\alpha^{t+1} - \alpha^t\|^2 \\
& + \langle \nabla_\alpha \mathcal{L}_p(u^t, \alpha^t) + \lambda \nabla_\alpha \mathcal{L}_a(w^t, \alpha^t) + \gamma \nabla_\alpha \mathcal{P}(\alpha^t), \alpha^{t+1} - \alpha^t \rangle.
\end{aligned}$$

Note that the β-terms cancel out. Substituting for the α-gradient from the iterations (2.12), we get:

$$\begin{aligned}
& \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t) \\
\leq\ & \big(-(\eta_u^t)^{-1} + L/2 + \beta/2 \big) \|u^{t+1} - u^t\|^2 + \big(-(\eta_w^t)^{-1} + \lambda L/2 + \beta/2 \big) \|w^{t+1} - w^t\|^2 \\
& + \big(-(\eta_\alpha^t)^{-1} + \frac{L(1+\lambda+\gamma)}{2} \big) \|\alpha^{t+1} - \alpha^t\|^2 \\
& + \langle \nabla_\alpha \mathcal{L}_p(u^t, \alpha^t) - \nabla_\alpha \mathcal{L}_p(u^{t+1}, \alpha^t), \alpha^{t+1} - \alpha^t \rangle \\
& + \lambda \langle \nabla_\alpha \mathcal{L}_a(w^t, \alpha^t) - \nabla_\alpha \mathcal{L}_a(w^{t+1}, \alpha^t), \alpha^{t+1} - \alpha^t \rangle. \qquad (2.17)
\end{aligned}$$

Using the Lipschitz gradient property (2.13), the last two inner product terms of (2.17) are upper bounded by:

$$\begin{aligned}
& L \|u^t - u^{t+1}\| \|\alpha^{t+1} - \alpha^t\| + L\lambda \|w^t - w^{t+1}\| \|\alpha^{t+1} - \alpha^t\| \\
\leq\ & \frac{L}{2} \big(\|u^t - u^{t+1}\|^2 + \|\alpha^{t+1} - \alpha^t\|^2 \big) + \frac{L\lambda}{2} \big(\|w^t - w^{t+1}\|^2 + \|\alpha^{t+1} - \alpha^t\|^2 \big).
\end{aligned}$$

It follows that:

$$\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t)$$

$$\leq \left[-(\eta_u^t)^{-1}+ L +\frac{\beta}{2}\right] \|u^{t+1}-u^t\|^2 + \left[-(\eta_w^t)^{-1}+ \lambda L +\frac{\beta}{2}\right] \|w^{t+1}-w^t\|^2$$
$$+ \left[-(\eta_\alpha^t)^{-1}+ (1+\lambda+\frac{\gamma}{2})L\right] \|\alpha^{t+1}-\alpha^t\|^2. \tag{2.18}$$

If there is a number T such that $(\eta_u^t, \eta_w^t, \eta_\alpha^t)$ satisfies the inequalities (2.15) for $t \geq T$, $\mathcal{L}(u^t, w^t, \alpha^t)$ is descending along the sequence (u^t, w^t, α^t). □

The following corollary is an immediate result of Theorem 2.2.

Corollary 2.3. *Suppose the conditions in Theorem 2.2 are satisfied. If the Lagrangian $\mathcal{L}$ has a lower bound, $\mathcal{L}(u^t, w^t, \alpha^t)$ converges to a number $\mathcal{L}^*$. Additionally, if $\mathcal{L}$ is coercive, the sequence (u^t, w^t, α^t) converges subsequentially to a critical point $(\bar{u}, \bar{w}, \bar{\alpha})$ of $\mathcal{L}(u, w, \alpha)$ obeying the equilibrium equations ($\gamma = 0$ for the case without regularization):*

$$\begin{aligned}
&\lambda \nabla_w \mathcal{L}_a(\bar{w}, \bar{\alpha}) + \beta(\bar{w} - \bar{u}) = 0,\\
&\nabla_u \mathcal{L}_p(\bar{u}, \bar{\alpha}) + \beta(\bar{u} - \bar{w}) = 0,\\
&\lambda \nabla_\alpha \mathcal{L}_a(\bar{w}, \bar{\alpha}) + \nabla_\alpha \mathcal{L}_p(\bar{u}, \bar{\alpha}) + \gamma\, \nabla_\alpha \mathcal{P}(\bar{\alpha}) = 0.
\end{aligned} \tag{2.19}$$

Proof. If the sequence $\mathcal{L}(u^t, w^t, \alpha^t)$ has a lower bound, it is clear that it is convergent, as we have proved that $\mathcal{L}(u^t, w^t, \alpha^t)$ is descending. For $c_4 = \frac{1}{2}\min\{c_1^{-1}, c_2^{-1}, c_3^{-1}\}$, it follows from (2.18) that:

$$\begin{aligned}
&c_4 \|(u^{t+1}-u^t, w^{t+1}-w^t, \alpha^{t+1}-\alpha^t)\|^2\\
\leq &\mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) \to 0,
\end{aligned} \tag{2.20}$$

as $t \to +\infty$, implying that

$$\lim_{t\to\infty} \|(u^{t+1}-u^t, w^{t+1}-w^t, \alpha^{t+1}-\alpha^t)\| = 0.$$

Since $\mathcal{L}$ is descending, lower bounded and coercive, $\|(u^t, w^t, \alpha^t)\|$ is uniformly bounded in t.

Then (u^t, w^t, α^t) subsequentially converges to a limit point $(\bar{u}, \bar{w}, \bar{\alpha})$. Let $(\eta_w^t, \eta_u^t, \eta_\alpha^t)$ tend to non-zero limit at large t, and take limits on both sides of equations (2.12). We obtain that $(\bar{u}, \bar{w}, \bar{\alpha})$ satisfies the equilibrium system (2.19). □

2.3.2 Comparisons with Other Gradient-based Methods

We present a simple analytical example from the DARTS paper [37] to see how first-order DARTS fails to find the optimum. Setting $\mathcal{L}_p = \mathcal{L}_2$ and $\mathcal{L}_a = \mathcal{L}_1$, we can write down explicitly the two-step iteration of first-order DARTS:

$$\begin{aligned} w^{t+1} &= w^t - \eta_w^t \nabla_w \mathcal{L}_a(w^t, \alpha^t) \\ \alpha^{t+1} &= \alpha^t - \eta_\alpha^t \nabla_\alpha \mathcal{L}_p(w^{t+1}, \alpha^t). \end{aligned} \tag{2.21}$$

Suppose (w^t, α^t) also converges subsequentially. Then we obtain the following equilibrium equations by taking the limits of equation (2.21):

$$\begin{aligned} &\nabla_w \mathcal{L}_a(\bar{w}, \bar{\alpha}) = 0 \\ &\nabla_\alpha \mathcal{L}_p(\bar{w}, \bar{\alpha}) = 0. \end{aligned} \tag{2.22}$$

Besides, we consider another gradient-based architecture search method named Mixed-Level NAS (MiLeNAS) [24], which includes $\mathcal{L}_a(w, \alpha)$ as a regularization term. The first-order MiLeNAS updates the pair (w, α) via:

$$\begin{aligned} w^{t+1} &= w^t - \eta_w^t \nabla_w \mathcal{L}_a(w^t, \alpha^t) \\ \alpha^{t+1} &= \alpha^t - \eta_\alpha^t \left(\nabla_\alpha \mathcal{L}_a(w^{t+1}, \alpha^t) + \lambda \nabla_\alpha \mathcal{L}_p(w^{t+1}, \alpha^t) \right), \end{aligned} \tag{2.23}$$

where λ is a non-zero hyperparameter to adjust the scale of the regularization term. Its idea of using both subsets of the dataset to update α is similar to RARTS, but MiLeNAS has missed the network splitting technique, which is the key of RARTS. Similarly, we derive the

equilibrium equations for first-order MiLeNAS:

$$\begin{aligned} &\nabla_w \mathcal{L}_a(\bar{w}, \bar{\alpha}) = 0 \\ &\nabla_\alpha \mathcal{L}_a(\bar{w}, \bar{\alpha}) + \lambda \nabla_\alpha \mathcal{L}_p(\bar{w}, \bar{\alpha}) = 0. \end{aligned} \tag{2.24}$$

Although the extra regularization term may improve the performance, its equilibria still suffer from the issue of incorrect limits. The following example compares the equilibria of these gradient-based methods for concrete objective functions, justifying the use of network splitting in RARTS.

Example 2.4. Suppose $\mathcal{L}_p(w, \alpha) = \alpha w - 2\alpha + 1$ and $\mathcal{L}_a(w, \alpha) = w^2 - 2\alpha w + \alpha^2$. The solution to the inner level problem of the bilevel problem (2.7) is:

$$w^*(\alpha) = \arg\min_w \mathcal{L}_a(w, \alpha) = \alpha,$$

since $w = \alpha$ minimizes the quadratic function $w^2 - 2\alpha w + \alpha^2$ given α fixed. After substitution, we obtain $\mathcal{L}_p(w^*(\alpha), \alpha) = \alpha^2 - 2\alpha + 1$. So the outer level problem becomes:

$$\min_\alpha \{\alpha^2 - 2\alpha + 1\},$$

with the global minimizer $(w^*, \alpha^*) = (1, 1)$. However, the equilibrium equations (2.22) of first-order DARTS indicate:

$$\begin{cases} 2\bar{w} - 2\bar{\alpha} = 0 \\ \bar{w} - 2 = 0, \end{cases}$$

which gives a spurious equilibrium $(\bar{w}, \bar{\alpha}) = (2, 2)$. The equilibrium equations (2.24) of

first-order MiLeNAS indicate:

$$\begin{cases} 2\bar{w} - 2\bar{\alpha} = 0 \\ 2\bar{\alpha} - 2\bar{w} + \lambda(\bar{w} - 2) = 0, \end{cases}$$

which gives the same equilibrium $(\bar{w}, \bar{\alpha}) = (2, 2)$. In this example, the regularization term of MiLeNAS does not improve the results, while network splitting of RARTS can.

We note that both loss functions $\mathcal{L}_a(w, \alpha)$ and $\mathcal{L}_p(u, \alpha)$ satisfy Lipschitz gradient property, implying descent of the Lagrangian $\mathcal{L}(u, w, \alpha)$ in equation (2.11) by Theorem 2.2, with $\gamma = 0$. Moreover, if $\lambda > 1/2$ and $\beta > 1$, we claim that the Lagrangian:

$$\mathcal{L}(u, w, \alpha) = \alpha u - 2\alpha + 1 + \lambda(w^2 - 2\alpha w + \alpha^2) + \frac{1}{2}\beta(u^2 - 2uw + w^2)$$

is coercive. Actually, we only need to consider the quadratic form:

$$\mathcal{S}(u, w, \alpha) = \alpha u + \lambda(w^2 - 2\alpha w + \alpha^2) + \frac{1}{2}\beta(u^2 - 2uw + w^2),$$

and its corresponding symmetric matrix:

$$\boldsymbol{A} = \begin{bmatrix} \lambda + \frac{1}{2}\beta & -\frac{1}{2}\beta & -\lambda \\ -\frac{1}{2}\beta & \frac{1}{2}\beta & \frac{1}{2} \\ -\lambda & \frac{1}{2} & \lambda \end{bmatrix}.$$

Note that $\boldsymbol{A}$ is positive definite and there is an orthogonal matrix $\boldsymbol{Q}$ such that $\boldsymbol{Q}^T \boldsymbol{P} \boldsymbol{Q} = \boldsymbol{A}$, where $\boldsymbol{P}$ is a diagonal matrix of all the eigenvalues of $\boldsymbol{A}$. Set $\boldsymbol{z} = (u, w, \alpha)^T$. We obtain:

$$\mathcal{S}(\boldsymbol{z}) = \boldsymbol{z}^T \boldsymbol{A} \boldsymbol{z} = (\boldsymbol{Q}\boldsymbol{z})^T \boldsymbol{P} \boldsymbol{Q}\boldsymbol{z} \geq \mu \|\boldsymbol{Q}\boldsymbol{z}\|^2 = \mu \|\boldsymbol{z}\|^2,$$

where $\mu > 0$ is the smallest eigenvalue of $\boldsymbol{A}$. This means that $\lim_{\|\boldsymbol{z}\| \to +\infty} \mathcal{S}(\boldsymbol{z}) = +\infty$. So

$\mathcal{L}(u, w, \alpha)$ is coercive. If we further require $\beta > 3/2$, $\mathcal{L}(u, w, \alpha)$ has a lower bound, since we have:

$$\mathcal{L}(u, w, \alpha) = \left[\alpha u + \lambda(w^2 - 2\alpha w) + (\lambda - \frac{1}{6})\alpha^2 + \frac{1}{2}\beta(u - w)^2\right] + \left[\frac{1}{6}\alpha^2 - 2\alpha + 1\right].$$

The first term in the brackets is a positive definite quadratic form, and the second term is a quadratic function which has a lower bound. So all the conditions of Corollary 2.3 are satisfied. The equilibrium equations (2.19) of RARTS indicate:

$$\begin{cases} \lambda(2\bar{w} - 2\bar{\alpha}) + \beta(\bar{w} - \bar{u}) = 0 \\ \bar{\alpha} + \beta(\bar{u} - \bar{w}) = 0 \\ \lambda(2\bar{\alpha} - 2\bar{w}) + (\bar{u} - 2) = 0, \end{cases}$$

which gives a unique solution when $4\beta\lambda - \beta - 2\lambda \neq 0$:

$$\begin{cases} \bar{w} = \frac{4\beta\lambda - 2\beta}{4\beta\lambda - \beta - 2\lambda} \\ \bar{u} = \frac{4\beta\lambda - 2\beta - 4\lambda}{4\beta\lambda - \beta - 2\lambda} \\ \bar{\alpha} = \frac{4\beta\lambda}{4\beta\lambda - \beta - 2\lambda}. \end{cases}$$

As $(w^*, \alpha^*) = (1, 1)$ is the global minimizer, the error is $O(\frac{1}{\lambda} + \frac{1}{\beta})$. We point out that one of the benefits of RARTS is the introducion of more freedom to adjust the hyperparameters λ and β via network splitting, so that the approximation can be further improved.

2.3.3 Rate of Convergence

We analyze the rate of convergence for Algorithm 1. Again, we list some definitions (see [41]) which help to establish the theorem.

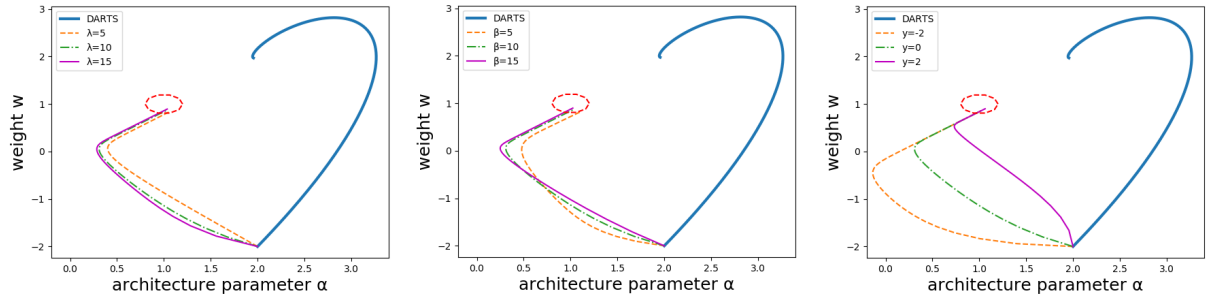

Figure 2.1: Learning trajectories of RARTS approach the global minimal point $(1, 1)$ of the solvable model at suitable values of λ, β and y_0 ($\lambda = 10$ in middle/right subplots, $\beta = 10$ in left/right subplots, $y_0 = 0$ in left/middle subplots), compared with that of the baseline (first-order DARTS).

(i) A differentiable function $f(x)$ defined on $\mathbb{R}^d$ is called *convex* if for $x, y \in \mathbb{R}^d$

$$f(x) - f(y) \leq \langle \nabla f(x), x - y \rangle. \tag{2.25}$$

(ii) A differentiable function $f(x)$ defined on $\mathbb{R}^d$ is called *strongly convex* if there is a number $\mu > 0$ so that for $x, y \in \mathbb{R}^d$

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu \|x - y\|^2. \tag{2.26}$$

We have the following lemma from [3]:

Lemma 2.5. *Suppose a differentiable function $f(x)$ is strongly convex and f^* is the minimum of $f(x)$. Then we have:*

$$f(x) - f^* \leq \frac{1}{2\mu} \|\nabla f(x)\|^2. \tag{2.27}$$

Proof. Set $g(t) = f(ty + (1 - t)x)$. We have

$$f(y) - f(x) = g(1) - g(0)$$

$$
\begin{aligned}
&= \int_0^1 g'(t)\,\mathrm{d}t \\
&= \int_0^1 \langle \nabla f(ty + (1-t)x), y - x\rangle\,\mathrm{d}t \\
&= \int_0^1 \langle \nabla f(ty + (1-t)x) - \nabla f(x), y - x\rangle\,\mathrm{d}t + \langle \nabla f(x), y - x\rangle \\
&\geq \int_0^1 \mu t\|y - x\|^2 \mathrm{d}t + \langle \nabla f(x), y - x\rangle \\
&= \langle \nabla f(x), y - x\rangle + \frac{\mu}{2}\|y - x\|^2,
\end{aligned}
$$

where the inequality is deduced from the fact that f is strongly convex. We note that the quadratic term $\langle \nabla f(x), y - x\rangle + \frac{\mu}{2}\|y - x\|^2$ is minimized when $\nabla f(x) + \mu(y - x) = 0$, and the minimum is $-\frac{1}{2\mu}\|\nabla f(x)\|^2$. Therefore, $f(y) - f(x) \geq -\frac{1}{2\mu}\|\nabla f(x)\|^2$. Note that this is true for any y. So $f^* - f(x) \geq -\frac{1}{2\mu}\|\nabla f(x)\|^2$. □

We have the following results on rate of convergence for Algorithm 1:

Corollary 2.6. *Suppose the conditions in Theorem 2.2 and Corollary 2.3 are satisfied. Suppose* $0 < \eta \leq \eta_u^t, \eta_w^t, \eta_\alpha^t \leq \frac{1}{3L}$ *for some* η.
(i) If the Lagrangian $\mathcal{L}(u, w, \alpha)$ *is strongly convex, we have linear convergence:*

$$
\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}^* \leq (1 - \frac{1}{3}c_4\eta^2\mu)(\mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}^*).
$$

(ii) If the Lagrangian $\mathcal{L}(u, w, \alpha)$ *is convex, and there is a number* M *such that* $\|(u^t, w^t, \alpha^t) - (\bar{u}, \bar{w}, \bar{\alpha})\| \leq M$*, we have sublinear convergence:*

$$
\mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}^* \leq \frac{6M^2}{c_4\eta^2 t}.
$$

Proof. It is clear that $\mathcal{L}(u, w, \alpha)$ also satisfies Lipschitz gradient property as all its component terms have Lipschitz gradients. Suppose the Lipschitz constant is also $L > 0$, without loss of generality. From the conclusion of Corollary 2.3, we obtain: $\mathcal{L}(\bar{u}, \bar{w}, \bar{\alpha}) = \mathcal{L}^*$. We derive

the bound from equation (2.20):

$$\begin{aligned}
& (\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}^*) - (\mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}^*) \\
= \;& \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t) \\
\leq \;& -c_4 \|(u^{t+1} - u^t, w^{t+1} - w^t, \alpha^{t+1} - \alpha^t)\|^2.
\end{aligned}$$

Using iteration (2.10), we have:

$$\|w^{t+1} - w^t\|^2 = (\eta_w^t)^2 \|\nabla_w \mathcal{L}(u^t, w^t, \alpha^t)\|^2. \tag{2.28}$$

Being aware of the inequalities $-\|A\|^2 + \frac{1}{2}\|B\|^2 \leq \|A + B\|^2 \leq 2\big(\|A\|^2 + \|B\|^2\big)$ for $A = \nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t) - \nabla_u \mathcal{L}(u^t, w^t, \alpha^t)$ and $B = \nabla_u \mathcal{L}(u^t, w^t, \alpha^t)$, we further have:

$$\begin{aligned}
\|u^{t+1} - u^t\|^2 &= (\eta_u^t)^2 \|\nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t)\|^2 \\
&\geq (\eta_u^t)^2 (-\|\nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t) - \nabla_u \mathcal{L}(u^t, w^t, \alpha^t)\|^2 + \frac{1}{2}\|\nabla_u \mathcal{L}(u^t, w^t, \alpha^t)\|^2) \\
&\geq (\eta_u^t)^2 (-L^2 \|w^{t+1} - w^t\|^2 + \frac{1}{2}\|\nabla_u \mathcal{L}(u^t, w^t, \alpha^t)\|^2) \\
&= (\eta_u^t)^2 \left[-L^2 (\eta_w^t)^2 \|\nabla_w \mathcal{L}(u^t, w^t, \alpha^t)\|^2 + \frac{1}{2}\|\nabla_u \mathcal{L}(u^t, w^t, \alpha^t)\|^2 \right] \\
&\geq -\frac{1}{9}(\eta_w^t)^2 \|\nabla_w \mathcal{L}(u^t, w^t, \alpha^t)\|^2 + \frac{1}{2}(\eta_u^t)^2 \|\nabla_u \mathcal{L}(u^t, w^t, \alpha^t)\|^2,
\end{aligned} \tag{2.29}$$

with the second inequality holds because of Lipschitz gradient property (2.13), and

$$\begin{aligned}\|u^{t+1}-u^t\|^2 &\le 2(\eta_u^t)^2(\|\nabla_u\mathcal{L}(u^t,w^{t+1},\alpha^t)-\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2)\\ &\le 2(\eta_u^t)^2(L^2\|w^{t+1}-w^t\|^2+\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2)\\ &= 2(\eta_u^t)^2(L^2(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2)\\ &\le \frac{2}{9}(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2+2(\eta_u^t)^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2.\end{aligned} \tag{2.30}$$

When deriving the above inequalities, we have also used the fact that $(\eta_u^t)^2L^2\le\frac{1}{9}$. Similarly, we have:

$$\begin{aligned}\|\alpha^{t+1}-\alpha^t\|^2 &= (\eta_\alpha^t)^2\|\nabla_\alpha\mathcal{L}(u^{t+1},w^{t+1},\alpha^t)\|^2\\ &\ge (\eta_\alpha^t)^2\big(-\|\nabla_\alpha\mathcal{L}(u^{t+1},w^{t+1},\alpha^t)-\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2\\ &\quad+\frac{1}{2}\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2\big)\\ &\ge (\eta_\alpha^t)^2\big[-L^2(\|u^{t+1}-u^t\|^2+\|w^{t+1}-w^t\|^2)+\frac{1}{2}\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2\big]\\ &\ge (\eta_\alpha^t)^2\big[-\frac{2}{9}(\eta_w^t)^2L^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2-2(\eta_u^t)^2L^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2\\ &\quad-(\eta_w^t)^2L^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\frac{1}{2}\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2\big]\\ &\ge -\frac{11}{81}(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2-\frac{2}{9}(\eta_u^t)^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2\\ &\quad+\frac{1}{2}(\eta_\alpha^t)^2\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2,\end{aligned} \tag{2.31}$$

using equation (2.28), inequality (2.30) and the fact that $(\eta_\alpha^t)^2L^2\le\frac{1}{9}$. Therefore, we have:

$$\begin{aligned}&(\mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1})-\mathcal{L}^*)-(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*)\\ \le\ & -c_4\big[(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2-\frac{1}{9}(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2\\ &+\frac{1}{2}(\eta_u^t)^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2-\frac{11}{81}(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2\end{aligned}$$

$$
\begin{aligned}
& -\frac{2}{9}(\eta_u^t)^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\frac{1}{2}\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2] \\
= \; & -c_4[\frac{61}{81}(\eta_w^t)^2\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\frac{5}{18}(\eta_u^t)^2\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2 \\
& +\frac{1}{2}(\eta_\alpha^t)^2\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2] \\
\leq \; & -\frac{1}{6}c_4\eta^2\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|^2, \qquad (2.32)
\end{aligned}
$$

using equation (2.28), inequalities (2.29) and (2.31) to obtain the first inequality, the fact that $\eta_u^t,\eta_w^t,\eta_\alpha^t\geq\eta$ to obtain the second inequalities. With strong convexity and Lemma 2.5, we have:

$$
(\mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1})-\mathcal{L}^*)-(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*)\leq-\frac{1}{3}c_4\eta^2\mu(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*),
$$

which implies:

$$
\mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1})-\mathcal{L}^*\leq(1-\frac{1}{3}c_4\eta^2\mu)(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*).
$$

If $\mathcal{L}(u^t,w^t,\alpha^t)$ is convex, we have:

$$
\begin{aligned}
\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^* &\leq\langle\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t),(u^t,w^t,\alpha^t)-(\bar{u},\bar{w},\bar{\alpha})\rangle \\
&\leq\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|\|(u^t,w^t,\alpha^t)-(\bar{u},\bar{w},\bar{\alpha})\| \\
&\leq M\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|,
\end{aligned}
$$

as we have proved that $\{u^t,w^t,\alpha^t\}$ is bounded uniformly on t. With inequality (2.32), this implies:

$$
\begin{aligned}
& (\mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1})-\mathcal{L}^*)-(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*) \\
\leq \; & -\frac{c_4\eta^2}{6M^2}(\mathcal{L}(u^t,w^t,\alpha^t)-\mathcal{L}^*)^2. \qquad (2.33)
\end{aligned}
$$

We set $\Delta_t = \mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}^*$, and have:

$$\frac{1}{\Delta_t} - \frac{1}{\Delta_{t+1}} = \frac{\Delta_{t+1} - \Delta_t}{\Delta_t \Delta_{t+1}} \leq -\frac{c_4 \eta^2}{6M^2} \frac{\Delta_t^2}{\Delta_t \Delta_{t+1}} \leq -\frac{c_4 \eta^2}{6M^2},$$

noting that Δ^t is descending. Therefore,

$$\frac{1}{\Delta_0} - \frac{1}{\Delta_t} = \sum_{i=0}^{t-1} \left(\frac{1}{\Delta_i} - \frac{1}{\Delta_{i+1}} \right) \leq -\frac{c_4 \eta^2 t}{6M^2},$$

which implies

$$\Delta_t = \mathcal{L}(u^t, w^t, \alpha^t) - \mathcal{L}^* \leq \frac{6M^2}{c_4 \eta^2 t}.$$

□

Example 2.7. Again, we consider $\mathcal{L}_p(w, \alpha) = \alpha w - 2\alpha + 1$ and $\mathcal{L}_a(w, \alpha) = w^2 - 2\alpha w + \alpha^2$. If $\lambda > 1/2$ and $\beta > \frac{3}{2}$, we claim that the Lagrangian:

$$\mathcal{L}(u, w, \alpha) = \alpha u - 2\alpha + 1 + \lambda(w^2 - 2\alpha w + \alpha^2) + \frac{1}{2}\beta(u^2 - 2uw + w^2)$$

is strongly convex. This is because the corresponding matrix of the quadratic form is positive definite, and hence,

$$\langle \nabla \mathcal{L}(\boldsymbol{z}_1) - \nabla \mathcal{L}(\boldsymbol{z}_2), \boldsymbol{z}_1 - \boldsymbol{z}_2 \rangle \geq 2\mu \|\boldsymbol{z}_1 - \boldsymbol{z}_2\|^2,$$

with $\boldsymbol{z}_i = (u_i, w_i, \alpha_i)^T$, where $\mu > 0$ is the smallest eigenvalue. Therefore, the Lagrangian converges linearly.

2.3.4 Search via SGD

To reduce the search cost of gradient descent in Algorithm 1, we can also use SGD with few modifications on iteration (2.12):

$$
\begin{aligned}
w^{t+1} &= w^t - \eta_w^t \lambda \frac{1}{m} \sum_{k=1}^{m} \nabla_w \mathcal{L}_{a,i_{t,k}}(w^t, \alpha^t) - \eta_w^t \beta (w^t - u^t) \\
u^{t+1} &= u^t - \eta_u^t \frac{1}{m} \sum_{k=1}^{m} \nabla_u \mathcal{L}_{p,i_{t,k}}(u^t, \alpha^t) - \eta_u^t \beta \, (u^t - w^{t+1}) \\
\alpha^{t+1} &= \alpha^t - \eta_\alpha^t \lambda \frac{1}{m} \sum_{k=1}^{m} \nabla_\alpha \mathcal{L}_{a,i_{t,k}}(w^{t+1}, \alpha^t) \\
&\quad - \eta_\alpha^t \frac{1}{m} \sum_{k=1}^{m} \nabla_\alpha \mathcal{L}_{p,i_{t,k}}(u^{t+1}, \alpha^t) - \eta_\alpha^t \gamma \, \nabla_\alpha \mathcal{P}(\alpha^t).
\end{aligned} \tag{2.34}
$$

Algorithm 2 summarizes the procedure for RARTS via SGD:

Algorithm 2: Relaxed Architecture Search via SGD

Input: number of epochs N, number of batches B, hyperparameters λ and β, a learning rate schedule $(\eta_w^t, \eta_u^t, \eta_\alpha^t)$, initialization of the weight parameters w^0, u^0 and the architecture parameters α^0.

Output: α^*, the architecture we want

Split the dataset $\mathcal{D}$ into two subsets $\mathcal{D}_p$ and $\mathcal{D}_a$. The initial time $t = 0$.

for $n = 0, 1, ..., N$ **do**

  for $b = 0, 1, ..., B$ **do**

    Sample a batch $\mathcal{D}_{p,t}$ from $\mathcal{D}_p$ and a batch $\mathcal{D}_{a,t}$ from $\mathcal{D}_a$ of size m uniformly (or cyclically). Compute $\mathcal{L}_{p,t}$ and $\mathcal{L}_{a,t}$ on $\mathcal{D}_{p,t}$ and $\mathcal{D}_{a,t}$ using equation (2.3) respectively, and then compute $\mathcal{L}_t$ by adding the two loss functions and the penalty terms in equation (2.11).

    Update the parameters via alternating SGD (see the expansion in iteration (2.34)):

    $w^{t+1} = w^t - \eta_w^t \nabla_w \mathcal{L}_t(u^t, w^t, \alpha^t)$

    $u^{t+1} = u^t - \eta_u^t \nabla_u \mathcal{L}_t(u^t, w^{t+1}, \alpha^t)$

    $\alpha^{t+1} = \alpha^t - \eta_\alpha^t \nabla_\alpha \mathcal{L}_t(u^{t+1}, w^{t+1}, \alpha^t)$

    $t = t + 1$

  end

end

We present some convergence results similar to Corollary 2.6. Unlike the gradient descent case, we need extra conditions on the variance of the gradient [2] in each SGD step.

Corollary 2.8. *Suppose the Lagrangian of each sample $\mathcal{L}_i(u, w, \alpha)$ satisfies the Lipschitz gradient property (2.13) for a constant $L > 0$ uniformly, and the averaged Lagrangian $\mathcal{L}(u, w, \alpha)$ is strongly convex and has a minimum $\mathcal{L}^*$. Suppose there are $M > 0$ and $C > 0$ such that:*

$$\mathbb{E}\left[\|\nabla_w \mathcal{L}_t(u^t, w^t, \alpha^t)\|^2 + \|\nabla_u \mathcal{L}_t(u^t, w^{t+1}, \alpha^t)\|^2 + \|\nabla_\alpha \mathcal{L}_t(u^{t+1}, w^{t+1}, \alpha^t)\|^2\right]$$

$$\leq \quad M + C\,\mathbb{E}\left[\|\nabla_{u,w,\alpha}\mathcal{L}(u^t, w^t, \alpha^t)\|^2\right]. \tag{2.35}$$

If $\max\{\eta_u^t, \eta_w^t, \eta_\alpha^t\} \leq K\eta$ *for some* K *and* $\eta \leq \frac{1}{4CLK^2}$, *and* $0 < \eta \leq \eta_u^t, \eta_w^t, \eta_\alpha^t \leq \frac{1}{3L}$ *when* t *is large, we have from the iteration in Algorithm 2:*

$$\mathbb{E}[\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1})] - \mathcal{L}^* \leq (1 - \frac{1}{2}\mu\eta)\left(\mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)] - \mathcal{L}^*\right) + MLK^2\eta^2, \tag{2.36}$$

and consequently (suppose the bounds on η *are always satisfied for* $t > 0$*)*

$$\mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)] - \mathcal{L}^* - \frac{2MLK^2\eta}{\mu} \leq (1 - \frac{1}{2}\mu\eta)^t\big(\mathbb{E}[\mathcal{L}(u^0, w^0, \alpha^0)] - \mathcal{L}^* - \frac{2MLK^2\eta}{\mu}\big). \tag{2.37}$$

Proof. Since $\mathcal{L}(u^t, w^t, \alpha^t)$ satisfies the Lipschitz gradient property, we have:

$$\begin{aligned}
& \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1}) - \mathcal{L}(u^t, w^t, \alpha^t) \\
\leq\ & \langle \nabla_w \mathcal{L}(u^t, w^t, \alpha^t), w^{t+1} - w^t\rangle + \langle \nabla_u \mathcal{L}(u^t, w^t, \alpha^t), u^{t+1} - u^t\rangle \\
& + \langle \nabla_\alpha \mathcal{L}(u^t, w^t, \alpha^t), \alpha^{t+1} - \alpha^t\rangle \\
& + \frac{L}{2}\left[\|w^{t+1} - w^t\|^2 + \|u^{t+1} - u^t\|^2 + \|\alpha^{t+1} - \alpha^t\|^2\right] \\
=\ & \langle \nabla_w \mathcal{L}(u^t, w^t, \alpha^t), -\eta_w^t\, \nabla_w \mathcal{L}_t(u^t, w^t, \alpha^t)\rangle + \langle \nabla_u \mathcal{L}(u^t, w^t, \alpha^t), -\eta_u^t\, \nabla_u \mathcal{L}_t(u^t, w^{t+1}, \alpha^t)\rangle \\
& + \langle \nabla_\alpha \mathcal{L}(u^t, w^t, \alpha^t), -\eta_\alpha^t\, \nabla_\alpha \mathcal{L}_t(u^{t+1}, w^{t+1}, \alpha^t)\rangle \\
& + \frac{L}{2}\left[\|w^{t+1} - w^t\|^2 + \|u^{t+1} - u^t\|^2 + \|\alpha^{t+1} - \alpha^t\|^2\right],
\end{aligned}$$

where we have used the three-step iteration of Algorithm 2 to get the first identity. We take expectations on both sides and use the law of total expectation:

$$\mathbb{E}[\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1})] - \mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)]$$

$$
\begin{aligned}
\leq\ & \mathbb{E}[-\eta_w^t\|\nabla_w\mathcal{L}(u^t,w^t,\alpha^t)\|^2]+\mathbb{E}[-\eta_u^t\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2]+\mathbb{E}[-\eta_\alpha^t\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2]\\
&+\mathbb{E}[\langle\nabla_u\mathcal{L}(u^t,w^t,\alpha^t),-\eta_u^t\left(\nabla_u\mathcal{L}_t(u^t,w^{t+1},\alpha^t)-\nabla_u\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\rangle]\\
&+\mathbb{E}[\langle\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t),-\eta_\alpha^t\left(\nabla_\alpha\mathcal{L}_t(u^{t+1},w^{t+1},\alpha^t)-\nabla_\alpha\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\rangle]\\
&+\frac{L}{2}\mathbb{E}\left[\|w^{t+1}-w^t\|^2+\|u^{t+1}-u^t\|^2+\|\alpha^{t+1}-\alpha^t\|^2\right].
\end{aligned}
$$

We note that

$$
\begin{aligned}
&\mathbb{E}[\langle\nabla_u\mathcal{L}(u^t,w^t,\alpha^t),-\eta_u^t\left(\nabla_u\mathcal{L}_t(u^t,w^{t+1},\alpha^t)-\nabla_u\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\rangle]\\
\leq\ & \eta_u^t\,\mathbb{E}\left[\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|\,\|\left(\nabla_u\mathcal{L}_t(u^t,w^{t+1},\alpha^t)-\nabla_u\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\|\right]\\
\leq\ & \frac{\eta_u^t}{2}\mathbb{E}\left[\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2+\|\left(\nabla_u\mathcal{L}_t(u^t,w^{t+1},\alpha^t)-\nabla_u\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\|^2\right]\\
\leq\ & \frac{\eta_u^t}{2}\mathbb{E}\left[\|\nabla_u\mathcal{L}(u^t,w^t,\alpha^t)\|^2\right]+\frac{\eta_u^t}{2}L^2\mathbb{E}\left[\|w^{t+1}-w^t\|^2\right],
\end{aligned}
$$

and similarly,

$$
\begin{aligned}
&\mathbb{E}[\langle\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t),-\eta_\alpha^t\left(\nabla_\alpha\mathcal{L}_t(u^{t+1},w^{t+1},\alpha^t)-\nabla_\alpha\mathcal{L}_t(u^t,w^t,\alpha^t)\right)\rangle]\\
\leq\ & \frac{\eta_\alpha^t}{2}\mathbb{E}\left[\|\nabla_\alpha\mathcal{L}(u^t,w^t,\alpha^t)\|^2\right]+\frac{\eta_\alpha^t}{2}L^2\mathbb{E}\left[\|w^{t+1}-w^t\|^2+\|u^{t+1}-u^t\|^2\right].
\end{aligned}
$$

Noting the fact that $0<\eta\leq\eta_u^t,\eta_w^t,\eta_\alpha^t\leq\frac{1}{3L}$ and $\max\{\eta_u^t,\eta_w^t,\eta_\alpha^t\}\leq K\eta$, we have:

$$
\begin{aligned}
&\mathbb{E}[\mathcal{L}(u^{t+1},w^{t+1},\alpha^{t+1})]-\mathbb{E}[\mathcal{L}(u^t,w^t,\alpha^t)]\\
\leq\ & \frac{-\eta}{2}\mathbb{E}\left[\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|^2\right]+L\mathbb{E}\left[\|w^{t+1}-w^t\|^2+\|u^{t+1}-u^t\|^2+\|\alpha^{t+1}-\alpha^t\|^2\right]\\
\leq\ & \frac{-\eta}{2}\mathbb{E}\left[\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|^2\right]+LK^2\eta^2(M+C\,\mathbb{E}\left[\|\nabla_{u,w,\alpha}\mathcal{L}(u^t,w^t,\alpha^t)\|^2\right]),
\end{aligned}
$$

since inequality (2.35) holds. With strong convexity and $\eta \leq \frac{1}{4CLK^2}$, this implies:

$$\begin{aligned} & \left(\mathbb{E}[\mathcal{L}(u^{t+1}, w^{t+1}, \alpha^{t+1})] - \mathcal{L}^*\right) - \left(\mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)] - \mathcal{L}^*\right) \\ \leq & -\mu(\eta - 2CLK^2\eta^2)\left(\mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)] - \mathcal{L}^*\right) + MLK^2\eta^2 \\ \leq & -\frac{1}{2}\mu\eta\left(\mathbb{E}[\mathcal{L}(u^t, w^t, \alpha^t)] - \mathcal{L}^*\right) + MLK^2\eta^2. \end{aligned}$$

Therefore, the bounds (2.36) and (2.37) are proved. □

Corollary 2.8 shows that when t is large and η is small, the expected Lagrangian $\mathbb{E}\left[\mathcal{L}(u^t, w^t, \alpha^t)\right]$ is very close to the minimum $\mathcal{L}^*$. However, a small initial learning rate can also lead to slow convergence. Previous research work [2] has shown a sublinear convergence when the learning rate is diminishing.

2.3.5 Regularization Terms

We explore the usage of the regularization term $\mathcal{P}(\alpha)$ in Lagrangian (2.11). First, we consider the *softmax function* [23], which are widely used to map a real valued vector $\boldsymbol{z} \in \mathbb{R}^d$ to a normalized vector in $[0, 1]^d$. For $\boldsymbol{z} = (z_1, ..., z_d)$, the softmax function $\sigma(\boldsymbol{z})$ is defined componentwise for $i = 1, 2, ..., d$:

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}.$$

Now we define the regularization function for a trainable vector $\boldsymbol{z}$ to be:

$$\mathcal{R}(\boldsymbol{z}) = \langle \sigma(\boldsymbol{z}), \boldsymbol{c} \rangle, \tag{2.38}$$

where $\boldsymbol{c} = (c_1, c_2, ..., c_d)$ is a vector of real numbers. We will discuss in the next chapter the meaning of $\boldsymbol{c}$. Here we point out that the architecture consists of several nodes, and each of them can be represented by a vector $\boldsymbol{z}_n$. Hence the architecture parameter α is the collection of all these vectors, and the overall regularization term $\mathcal{P}(\alpha)$ is the sum of such $\mathcal{R}(\boldsymbol{z}_n)$. The following proposition shows that the gradient of $\mathcal{P}(\alpha)$ is Lipschitz, and hence satisfies the condition of Theorem 2.2.

Proposition 2.9. *The regularization function* $\mathcal{P}(\alpha) = \sum_{n=1}^{N} \mathcal{R}(\mathbf{z}_n)$ *satisfies the Lipschitz gradient property (2.13).*

Proof. We only need to show that $\mathcal{R}(\boldsymbol{z}_n)$ has Lipschitz continuous gradients. That means, we only need to show that each component of $\sigma(\boldsymbol{z})$ has Lipschitz continuous gradients (we have dropped the subscript). This can be implied by the boundness of the second order derivatives. Actually, we have:

$$\partial_k(\sigma(\boldsymbol{z})_i) = \begin{cases} -\sigma(\boldsymbol{z})_i\sigma(\boldsymbol{z})_k, & k \neq i \\ \sigma(\boldsymbol{z})_i - \sigma(\boldsymbol{z})_i^2, & k = i. \end{cases}$$

Applying chain rule and the fact that $0 \leq \sigma(\boldsymbol{z})_i \leq 1$, we can easily show that all the second order derivatives are uniformly bounded. □

Chapter 3

Search for Topological Architectures of CNN Blocks via RARTS

Since we have illustrated the relaxed search algorithm in the previous chapter, we now present the formulation of the architecture search problem for CNN blocks, which includes the search spaces, selection criteria, constraints and evaluation. We discuss in detail the image classification task, the outline of various datasets, and the hyperparameter and device settings to be used in the experiments. Comparisons are made among various architecture search methods through experimental results.

3.1 Problem Formulation

3.1.1 Basic Operations

We shall describe the basic operations of a Convolutional Neural Network (CNN) [23].

Convolution. Since CNN is widely used to extract features from images, the input of an

operation or a *layer* is usually an image or *feature map*, which can be represented by a two-dimensional (2D) tensor $\boldsymbol{X} \in \mathbb{R}^{H\times W}$. Here H and W are the *height* and *width* of the input feature. The 2D *convolution* operation can be interpreted as a shifting dot product between the input feature and the *kernel*, which can also be represented by a 2D tensor $\boldsymbol{K} \in \mathbb{R}^{k\times k}$. Here k is the *kernel size*, which is often set to be 3, 5 or 7 in many CNN architectures. A 2D convolution is defined to be a map f, mapping $\boldsymbol{X} \in \mathbb{R}^{H\times W}$ to a 2D output feature tensor $\boldsymbol{Y} = f(\boldsymbol{X}) = \boldsymbol{X} * \boldsymbol{K}$, so that for $1 \le h \le H$ and $1 \le w \le W$ we have elementwise:

$$\boldsymbol{Y}^{h,w} = \sum_{1\le r,s\le k} \boldsymbol{X}^{h+r,w+s}\boldsymbol{K}^{r,s}. \tag{3.1}$$

Here we can require appropriate zero *padding* (so that the dimension of $\boldsymbol{X}$ is expanded) and the *stride* to be 1, so that the dimension of the output feature is the same as that of the input feature, i.e., $\boldsymbol{Y} \in \mathbb{R}^{H\times W}$.

Suppose we have a batch of images or feature maps with multiple channels, represented by a 4D tensor $\boldsymbol{X} \in \mathbb{R}^{N\times C_{in}\times H\times W}$. Here N is the *batch size* and C_{in} is the number of *input channels*, e.g., $C_{in} = 3$ for initial input images of three colors. In this case, the kernels can also be represented by a 4D tensor $\boldsymbol{K} \in \mathbb{R}^{C_{out}\times C_{in}\times k\times k}$, where C_{out} is the number of *output channels*. If we require appropriate zero padding and the stride to be 1, the 2D convolution f maps $\boldsymbol{X} \in \mathbb{R}^{N\times C_{in}\times H\times W}$ to $\boldsymbol{Y} = f(\boldsymbol{X}) \in \mathbb{R}^{N\times C_{out}\times H\times W}$. The output features $\boldsymbol{Y}$ can be defined componentwise for $1 \le n \le N$ and $1 \le j \le C_{out}$:

$$\boldsymbol{Y}^{nj} = \sum_{i=1}^{C_{in}} \boldsymbol{X}^{ni} * \boldsymbol{K}^{ji}, \tag{3.2}$$

for the version without the *bias* terms. Here $\boldsymbol{Y}^{nj} \in \mathbb{R}^{1\times 1\times H\times W}$ is the feature map of the n-th batch and the j-th output channel, while $\boldsymbol{X}^{ni} \in \mathbb{R}^{1\times 1\times H\times W}$ and $\boldsymbol{K}^{ji} \in \mathbb{R}^{1\times 1\times k\times k}$ have similar meanings.

Batch normalization. A convolution is sometimes followed by a *batch normalization* [30],

which can normalize the input features by adjusting the mean and variance of a batch. Suppose $\boldsymbol{X} \in \mathbb{R}^{N\times C_{in}\times H\times W}$ stands for the input features, $\boldsymbol{X}^i \in \mathbb{R}^{N\times 1\times H\times W}$ is a batch of feature maps for the i-th channel, and x^{nihw} is an element of $\boldsymbol{X}^i$. It is clear that $\boldsymbol{X}^i$ has NHW such elements. The batch mean $\mu^i_{\mathcal{B}}$ and batch variance $(\sigma^i_{\mathcal{B}})^2$ of the i-th channel are defined for $1 \le i \le C_{in}$ as follows:

$$\mu^i_{\mathcal{B}} = \frac{1}{NHW}\sum_{n,h,w} x^{nihw}$$
$$(\sigma^i_{\mathcal{B}})^2 = \frac{1}{NHW}\sum_{n,h,w} (x^{nihw} - \mu^i_{\mathcal{B}})^2.$$

The normalized feature maps of the i-th channel $\widehat{\boldsymbol{X}}^i$ are defined elementwise:

$$\widehat{x}^{nihw} = \frac{x^{nihw} - \mu^i_{\mathcal{B}}}{\sqrt{(\sigma^i_{\mathcal{B}})^2 + \epsilon}},$$

where ϵ is a constant number. By introducing a pair of learnable parameters $\gamma = (\gamma^1, ..., \gamma^{C_{in}})$ and $\beta = (\beta^1, ..., \beta^{C_{in}})$, the batch normalization operation transforms $\boldsymbol{X} \in \mathbb{R}^{N\times C_{in}\times H\times W}$ to a tensor $\boldsymbol{Y}$ of the same shape, defined elementwise:

$$y^{nihw} = \gamma^i\, \widehat{x}^{nihw} + \beta^i. \tag{3.3}$$

Batch normalization is believed to be a way to speed up the training and regularize the model [30], while it has also been pointed out that the loss function can be smoother with the introduction of batch normalization [47].

ReLU. *Activation functions* contribute to the nonlinearity of CNNs. Among them, *ReLU* (Rectified Linear Unit) [21] is commonly used after a convolution or batch normalization. For the input features $\boldsymbol{X} \in \mathbb{R}^{N\times C_{in}\times H\times W}$, the output of the ReLU function σ is a tensor

$\boldsymbol{Y} = \sigma(\boldsymbol{X}) \in \mathbb{R}^{N \times C_{in} \times H \times W}$, which can be defined elementwise:

$$y^{nihw} = \max\{x^{nihw}, 0\}.$$

We shall note that ReLU does not change the resolution $H \times W$ of the input feature maps, while batch normalization or convolution with stride 1 and proper padding does not, either.

Pooling. If we want to produce feature maps of different resolutions, we may use convolutions with strides greater than 1, or simply *pooling layers* [23]. The *max pooling* maps each 2×2 patch of the input features to the maximum of its 4 elements. In this way, the resolution of the output features is reduced to $\frac{H}{2} \times \frac{W}{2}$. Similarly, one can define *average pooling* to be a function mapping a patch to the average of its elements.

Shortcut and identity. The convolution, batch normalization, activation and pooling operations are often stacked sequentially to build many well-known CNNs [33, 32, 48]. However, it is believed that these sequential *plain* architectures might not be easy to train when they become deeper, and hence a residual architecture called *ResNet* has been proposed [26]. ResNet has added a few *shortcuts* to the sequential architecture, and each shortcut is set to be a *skip connection* or *identity* operation (i.e., it maps $\boldsymbol{X}$ to $\boldsymbol{X}$). In this way, the output feature map $\boldsymbol{Y}$ of an operation f plus the shortcut is:

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \boldsymbol{X},$$

instead of $\boldsymbol{Y} = f(\boldsymbol{X})$ in the plain architectures. Since $f(\boldsymbol{X}) = \boldsymbol{Y} - \boldsymbol{X}$, the target f which we would like to train, is a *residual* mapping. It has been pointed out that the deeper model can at least outperform the shallower model since it can degrade to the shallower one if the residuals are learned to be 0, i.e., the extra layers are identities. The introduction of shortcuts has greatly enriched the topological architecture of CNNs, extending beyond the plain ones.

Depthwise separable convolution. We have reviewed the elementary operations of CNNs. Now we construct more complicated variants based on these building bricks. A *depthwise separable convolution* splits the convolution operation defined in equation (3.2) into a *depthwise convolution* and a *pointwise convolution*, for the purpose of reducing computational complexity [29]. Suppose $\boldsymbol{X} \in \mathbb{R}^{N \times C_{in} \times H \times W}$ is the input feature map, and stride and padding are chosen so that the output resolution is the same as the input resolution. The depthwise convolution only requires C_{in} kernels, and can be represented by a tensor $\boldsymbol{K}_d \in \mathbb{R}^{1 \times C_{in} \times k \times k}$. Each output feature is only associated with one input feature and one kernel of the corresponding channel. It generates the intermediate feature maps $\widehat{\boldsymbol{X}} \in \mathbb{R}^{N \times C_{in} \times H \times W}$:

$$\widehat{\boldsymbol{X}}^{ni} = \boldsymbol{X}^{ni} * \boldsymbol{K}_d^i,$$

for $1 \leq n \leq N$ and $1 \leq i \leq C_{int}$. Here $\boldsymbol{K}_d^i \in \mathbb{R}^{1 \times 1 \times k \times k}$ is the kernel of the i-th channel. After that, the pointwise convolution is applied to the intermediate features to generate output features with C_{out} channels. The pointwise convolution can be represented by $\boldsymbol{K}_p \in \mathbb{R}^{C_{out} \times C_{in} \times 1 \times 1}$, while the output features $\boldsymbol{Y} \in \mathbb{R}^{N \times C_{out} \times H \times W}$ can be defined componentwise by the linear transform:

$$\boldsymbol{Y}^{nj} = \sum_i \widehat{\boldsymbol{X}}^{ni} k_p^{ji}.$$

We note that the computational cost of the convolution of a kernel with shape $k \times k$ and a feature with shape $H \times W$ is $O(k^2 HW)$. So the regular convolution with C_{in} input channels and C_{out} output channels (3.2) has a parameter number of $O(C_{in} C_{out} k^2)$ and a computational cost of $O(C_{in} C_{out} k^2 HW)$, if we treat the batch size N as a constant (or simply 1). As the combination of the depthwise convolution and the pointwise convolution, the depthwise separable convolution has a parameter number of $O(C_{in} k^2 + C_{in} C_{out})$ and a computational cost of $O(C_{in} k^2 HW + C_{in} C_{out} HW)$. It is clear that the depthwise separable convolution can reduce the complexity to $1/k^2$ of the regular convolution, if $C_{out} >> k^2$.

Dilated convolution. Another commonly used variant of convolution is called the *dilated convolution* [62]. While each output pixel is associated with a square of adjacent input pixels for the regular convolution, there is a jump between the corresponding input pixels for the dilated convolution, which is defined to be the *dilation factor*. For an input feature $\boldsymbol{X} \in \mathbb{R}^{H \times W}$ and a kernel $\boldsymbol{K} \in \mathbb{R}^{k \times k}$, the output of the dilated convolution of a dilation factor d is defined to be $\boldsymbol{Y}$, where:

$$\boldsymbol{Y}^{h,w} = \sum_{1 \le r,s \le k} \boldsymbol{X}^{h+dr,w+ds} \boldsymbol{K}^{r,s}$$

for $1 \le h \le H$ and $1 \le w \le W$ with suitable padding. This definition can be extended to the convolution with multiple input and output channels using equation (3.2). We shall note that when $d = 1$, the dilated convolution is the same as the regular convolution. It has been pointed out that dilated convolution [62] can increase the *receptive field*, which is known to be the input patch or pixels that contribute to the value of the output patch or pixel via convolution [23]. For example, the receptive field of an output pixel generated by a 3×3 regular convolution is a 3×3 input patch, and the receptive field of this 3×3 patch is a 5×5 patch, given another layer of 3×3 regular convolution. For a dilated convolution with the kernel of the same shape, the receptive field of the 3×3 patch is a 7×7 patch, if the dilation factor $d = 2$. A larger receptive field means that a pixel can be indirectly connected to more pixels in previous layers. *Downsampling* methods like pooling or striding are often adopted to increase the receptive field so that the model can learn more global features for tasks like *image classification*. However, downsampling reduces the resolution of the features, and hence might affect the performance of the model on more complicated tasks like *semantic segmentation*, which needs to recover the full-resolution output. Dilated convolution is believed to be better than these downsampling methods, as it can preserve the resolution [62].

3.1.2 Search Spaces

Since shortcuts are introduced, we need to search not only the type of the operations, but also the *edges* between the states, i.e., the topology of the neural network. It would be computationally expensive if one wants to search for specific types of all the operations and the edges to connect the states. As many successful architectures are built of repeated modules of a certain type and function [49, 26], a simple search space called the *NASNet search space* is composed with the help of *cells*, i.e., the repeated modules of the same structure [70]. Architecture search is made easy as there are only two cell structures needed to be searched for, which are named *normal cell* and *reduction cell*. The normal cell can preserve the resolution of the input features, while the reduction cell decreases the resolution by a half. To construct a CNN, the cells are stacked in the way that $3N$ normal cells are piled consecutively, with two reduction cells inserted after the N-th and the $2N$-th normal cells. The input and output features of cells are called *hidden states*. The $(i+1)$-th cell has two input hidden states from the $(i-1)$-th and the i-th cells, and the output of the $(i+1)$-th cell is the input hidden state of the $(i+2)$-th and $(i+3)$-th cell. Given all these global architectures fixed, one needs to search for the local architectures, namely, the type of operations and the edges between states within the normal and reduction cells.

Suppose there are 4 intermediate states within each cell. Along with two input hidden states and one output hidden state, we need to search edges between seven states. Although this search space is much smaller than a global one, it is still computationally expensive to train all the models selected to convergence in the search process, even if an order of the intermediate states is predefined [70, 36, 43]. To solve this issue, a *supernet* structure and a training technique called *weight sharing* have been proposed [42, 37]. The supernet is a *directed acyclic graph*, containing all the candidate edges, and each of its subgraphs can represent an architecture. To search for an architecture, we only need to search for a subgraph. Weight sharing means that we only need to train the supernet once, and each architecture can

take the weights directly from the trained supernet, as it is a part of the supernet. It has created more space for many Differentiabl Neural Architecture Search (DNAS) methods like DARTS [37] and RARTS, as the differentiable architecture parameters have to be learned on the supernet.

For a cell of DARTS, the four intermediate states are labeled by four numbers $0-3$ [37]. The features or information can only flow from the input states or a state with a smaller label to a state with a larger label, e.g., the input of the 0-th state is the sum of the two edges connecting the two input hidden states, while the input of the 2-nd state is the sum of the four edges connecting the two input hidden states and the 0-th and 1-st states. Therefore, we have obtained $2 + 3 + 4 + 5 = 14$ candidates edges in a cell of the supernet. For each intermediate state, we need to select two optimal edges, which means we need to select 8 edges out of the 14 candidates to build an architecture from the supernet. Finally, the output of the cell is a concatenation of the outputs from the four intermediate states, along the channel dimension. It serves as the input hidden state of the next two cells. Everything in the concatenation step is fixed.

When selecting the edges, we also need to learn an operation for each edge. The set of candidate operations is predefined, among which are often identity, zero, max or average pooling, convolution, depthwise separable or dilated convolution of different kernel sizes and strides, or their combinations [70, 42, 36, 43, 37]. There are often 7 or 8 candidate operations in the search space of many DNAS methods [37, 5].

3.1.3 Selection Criteria

Suppose $\boldsymbol{X}_i \in \mathbb{R}^{N \times C_{in} \times H \times W}$ is the i-th state, $O_{i,j,k}$ is the k-th candidate operation of the edge from the i-th state to the j-th state for $i < j$ and $1 \leq k \leq K$, where K is the number of

candidate operations. The j-th state is the sum of the outputs from all the previous states:

$$\boldsymbol{X}_j = \sum_{i<j} O_{i,j}(\boldsymbol{X}_i), \tag{3.4}$$

where $O_{i,j}$ is the optimal operation for edge (i, j), which can be further written as:

$$O_{i,j} = \sum_k c_{i,j,k} O_{i,j,k}. \tag{3.5}$$

Here $\boldsymbol{c}_{i,j} = (c_{i,j,1}, ..., c_{i,j,K})$ is the one-hot encoding for the optimal candidate operation $O_{i,j,l}$, i.e., $c_{i,j,k} = \delta^{kl}_{i,j}$, for δ^{kl} the Kronecker delta function.

As reinforcement learning and evolutionary algorithms [70, 42, 43] are believed to be less efficient [37], DARTS has developed a gradient-based algorithm to learn the architecture, which has been discussed exhaustively in the previous chapter. To apply gradient descent, the binary structure in equation (3.5), i.e., selecting an operation or not, has to be transformed to continuous architecture parameters. With the architecture parameter $\boldsymbol{\alpha}_{i,j} = (\alpha_{i,j,1}, ..., \alpha_{i,j,K}) \in \mathbb{R}^K$, a mixed operation along edge (i, j) with input $\boldsymbol{X}_i$ is set up via continuous relaxation of equation (3.5) [37]:

$$M_{i,j} = \sum_k \sigma(\boldsymbol{\alpha}_{i,j})_k O_{i,j,k}, \tag{3.6}$$

where σ is the softmax function we have discussed in Section 2.3.5, which is used to normalize a real-valued vector to a probability distribution. During training, $M_{i,j}$ is used to replace $O_{i,j}$ to compute $\boldsymbol{X}_j$ in equation (3.4). Therefore, α is included in the computation of the loss function and can be learned by many gradient-based algorithms. Suppose there are 8 candidate operations, i.e. $K = 8$. Then there are $8 \times 14 = 112$ architecture parameters in one cell, which means we need to learn the values of $112 \times 2 = 224$ architecture parameters since there are two kinds of cells.

After training, we need to select the optimal operations. Clearly, the value $\sigma(\boldsymbol{\alpha}_{i,j})_k$ can be interpreted as the probability or the *score* of the operation $O_{i,j,k}$, and thus the optimal operation for edge (i, j) can be chosen via:

$$O_{i,j} = O_{i,j,l}\ ,\ l = l(i,j) = \arg\max_{1 \le k \le K} \sigma(\boldsymbol{\alpha}_{i,j})_k.$$

In addition to the search of operations, DARTS also searches for the edges by restricting the number of input edges to be 2 [37]. That is to say, for maximum operation probabilities over edges $\{\sigma(\boldsymbol{\alpha}_{i,j})_l\}_{i<j}$, we need to further select two input states i_1 and i_2 with $i_1 \neq i_2$ such that $\sigma(\boldsymbol{\alpha}_{i_1,j})_l$ and $\sigma(\boldsymbol{\alpha}_{i_2,j})_l$ are the largest and second largest. Finally, the j-th state of the selected architecture is:

$$\boldsymbol{X}_j = \sum_{i=i_1,i_2} O_{i,j}(\boldsymbol{X}_i).$$

We should note that it is only a portion of equation (3.4) for the supernet, as only two edges are selected.

3.1.4 Constraints on Model Efficiency

We sometimes impose constraints of model efficiency on complicated models like Deep Neural Networks (DNNs), when the computational resources are restricted. These constraints limit the model size, FLOPS or *latency* (inference time), and thus the constrained models are usually smaller but more efficient than the full-size models. There are many manually designed efficient models [29], while there are also some approaches to learn the efficient models from the given full-size models. From the view of architecture search, this process can be formulated as searching for an efficient architecture from the given supernet. To automate the selection towards an efficient architecture, the constraints are often quantified

as a function and added to the model loss as a penalty. When minimizing to penalized loss, the algorithm is likely to converge to a more efficient architecture. That is why one may set up the regularized Lagrangian (2.11) from the original Lagrangian (2.8).

Since we want to use the RARTS algorithm to minimize the regularized Lagrangian (2.11), we may want to specify the penalty function $\mathcal{P}$. There are many options, including the ℓ_0 or ℓ_1 norm on parameters or groups of parameters [40, 63]. The ℓ_0 norm is equal to the number of parameters, and hence the model size will be reduced if the ℓ_0 norm is small. The ℓ_1 norm is able to generate a *sparse* network, i.e., a large number of parameters or parameters groups are 0. Moreover, the group sparsity is able to produce light models [59, 15, 4], combined with a network compression technique called *pruning*, which we are going to discuss in detail in the next chapter.

However, these norms act directly on the weight parameters to reduce the model size, and does not correctly reflect the model latency, which is essential to many real-world applications. It has been pointed out that two operations can have the same number of parameters but their latencies differ by 60% [9]. Since we want to search for the architecture of more efficiency, the latency of its operations should be taken into account, as the real latency of the architecture can be approximated by the sum of the latencies from all its operations. While the latency for each operation can be measured via PyTorch/TensorRT, it depends on what kind of devices we use. Suppose the device is fixed during the search stage, and $LAT_{i,j,k}$ is the latency of the k-th operation on the (i,j)-th edge. Then the latency of the selected operation on the (i,j)-th edge is:

$$LAT_{i,j} = \sum_k c_{i,j,k} LAT_{i,j,k},$$

and analogous to the definition of the mixed operation in equation (3.6), a continuous relax-

ation has been proposed [9]:

$$MLAT_{i,j} = \sum_k \sigma(\boldsymbol{\alpha}_{i,j})_k LAT_{i,j,k},$$

and the penalty $\mathcal{P}(\alpha)$ is defined to be the sum of all the $MLAT_{i,j}$. This definition is consistent with our discussions in Section 2.3.5, and we shall be aware that the penalty depends only on α.

3.2 Experiments

3.2.1 Datasets and Settings

We go over how convolutional layers or cells are used to solve image classification problems. The task of image classification is to assign each image a label, which is an integer. Suppose there are $C = 10$ image classes represented by numbers from 0 to 9, the correct label for the n-th image is y_n, and the predicted label is $\hat{y}_n$. Then y_n and $\hat{y}_n$ must be integers between 0 and 9. As the output of the cumulative convolutional cells is a feature map $\boldsymbol{X} \in \mathbb{R}^{N \times C_{out} \times H \times W}$, further transforms have to be taken. A pooling layer is usually applied to downsample each feature of size $H \times W$ to a single point, and obtain a batch of N vectors: $\boldsymbol{V} \in \mathbb{R}^{N \times C_{out}}$. These vectors are then transformed by a *fully connected layer* [33], which is a learnable linear map represented by a matrix $\boldsymbol{T} \in \mathbb{R}^{C_{out} \times C}$, and the obtain the output $\boldsymbol{U} \in \mathbb{R}^{N \times C}$ via matrix multiplication: $\boldsymbol{U} = \boldsymbol{V}\,\boldsymbol{T}$. After that, a softmax function is applied to each vector of $\boldsymbol{U}$, and obtain a distribution for the predicted labels: $\boldsymbol{D} = \sigma(\boldsymbol{U}) \in \mathbb{R}^{N \times C}$. Here $\boldsymbol{D}$ contains a batch of N vectors, each has length C and represents the probabilities of the C classes. During evaluation or inference, the predicted label for the n-th image is $\hat{y}_n = l$, if the l-th element of $\boldsymbol{D}^n$ is the largest, i.e., $l = \arg\max_k \boldsymbol{D}^{n,k}$. During training, we use the cross-entropy function defined in by equation (1.2) to compute the loss, e.g., the first term or the

second term without λ in equation (2.11):

$$\mathcal{L} = \frac{1}{N}\sum_{n} H_o(p^n, q^n) = \frac{1}{N}\sum_{n} H_o(p^n, \boldsymbol{D}^n) = -\frac{1}{N}\sum_{n} \log \boldsymbol{D}^{n,y_n}.$$

Then the loss is minimized by a gradient-based method like RARTS so that the parameters could be learned.

We first consider searching for the architecture (the normal and reduction cells) via RARTS on the CIFAR-10 dataset [31], which is a common dataset for image classification, consisting of 50,000 images for training and 10,000 images for testing. The images are colored and have 32×32 resolutions, and thus the input tensor of the whole model (without data augmentation) should belong to $\mathbb{R}^{N\times 3\times 32\times 32}$, where N is the batch size. The 50,000 training images are allocated to 10 object classes evenly, with 5,000 images in each class. When training the search stage, the architecture parameters are learned by RARTS and the architecture is selected according to the criterion we have described in Section 3.1.3, where $\mathcal{D}_p$ constains half of the original training data and $\mathcal{D}_a$ contains another half as described in the data splitting section. Subsequently, the selected architecture is trained again from scratch, and evaluated on CIFAR-10. The criterion for evaluating the architecture is the test accuracy, i.e., the percentage of test images' labels which are predicted correctly. The settings of hyperparameters are similar to that of DARTS [37]. The network to be searched for is built of 8 cells with 6 normal cells and 2 reduction cells, arranged in the order described in Section 3.1.2. For the search stage, batch size = 64, initial weight learning rate = 0.025, momentum = 0.9, weight decay = 0.0003, initial alpha learning rate = 0.0003, alpha weight decay = 0.001, epochs = 50. For the retraining stage, batch size = 96, learning rate = 0.025, momentum = 0.9, weight decay = 0.0003, epochs = 600.

In addition, we also evaluate the architecture on the ImageNet-1000 dataset [14, 45], which contains 1,281,167 training images and 50,000 validation images, divided into 1,000 object

classes. These images are also colored but vary in resolutions. The input images are usually resized to a resolution of 224×224, so that the input tensor belongs to $\mathbb{R}^{N\times3\times224\times224}$. The larger resolution and number of image classes may require a more complicated model to learn the features. Along with the large volume of the dataset, it has made the direct architecture search on ImageNet-1000 computationally expensive. We follow the trick of *transfer learning* [70, 37], namely searching for a proxy architecture on a smaller dataset like CIFAR and then transferring to ImageNet-1000 for retraining and evalution. To be specific, the cells learned on CIFAR-10 are stacked to get a network consisting of 14 cells, with 12 normal ones and 2 reduction ones, to be retrained on ImageNet-1000. For this retraining stage, batch size = 128, learning rate = 0.1, momentum = 0.9, weight decay = 0.00003, epochs = 250. As transfer learning is developed for the purpose of reducing computations, it is not always necessary. There are many other differentiable methods which can search for the architecture on ImageNet-1000 directly. ProxylessNAS [5] has proposed keeping only one operation of each edge active during training so that the memory is occupied by a compact model only, instead of the supernet of all the candidate operations. It has further enlarged the diversity of the learned architecture as there is no more need to stack cells of the same structures for retraining on ImageNet.

We also consider searching the architecture from NATS-Bench [17], which is a benchmark of search space other than that of DARTS, for comparing different NAS methods. The benefit of using such a benchmark is that all the candidate architectures in its search space have already been evaluated under the same settings, and there is no need to optimize the hyperparameters. This consistent setting has made the comparisons between different NAS methods fairer, whereas previous NAS methods are often evaluated in different search spaces, with different training tricks, so that it is hard to know whether the improvement comes from the new method or the training tricks. Although the distribution of the normal and reduction cells is the same as that of DARTS, NATS-Bench has 4 nodes in each cell and 5 candidate operations for each edge, which are less than those of DARTS. Overall, NATS-Bench has

provided and evaluated 15,625 candidate architectures. We apply RARTS to search the architectures from this search space, on multiple datasets like CIFAR-10 and CIFAR-100 [31]. CIFAR-100 has the same volume and resolution of images as those of CIFAR-10, while its images are divided evenly into 100 object classes, which means each class only contains 500 images for training and 100 images for testing. The hyperparameters for NATS-Bench are similar to those of DARTS, except for a weight decay of 0.0005, and a learning rate decaying from 0.1 to 0 during the retraining stage [17].

3.2.2 Results

We search the architecture on CIFAR-10 with RARTS, following the search space settings, selection criterion and the latency constraint discussed in the previous section. The latency regularization term is scaled by a hyperparameter γ as shown in equation (2.11) so that it is balanced with other loss terms. Typically, if we increase the latency scale, the model we find will be smaller in size. For the current search, γ is set to be 0.002 so that the model size can be comparable to those in prior works. The discovered architecture is then evaluated on CIFAR-10, with the experimental results shown in Table 3.1. The model of 3.2M parameters with 2.65% error on the test data is more accurate than the 3.3M model of 3.00% error found by first-order DARTS and the 3.3M model of 2.76% error found by second-order DARTS. Apart from the higher accuracy, the model found by RARTS is more stable in that it has low variance in the average accuracy of 5 runs. Moreover, RARTS has achieved better search efficiency than the second-order DARTS and has reduced the search time by around 60%. Finally, we point out that the performance of the model found by RARTS is comparable to all the other differentiable methods listed in the table. One may note that ProxylessNAS has obtained the highest accuracy, but we have mentioned that it has adopted a larger search space and its resulting model of 5.7M is also larger.

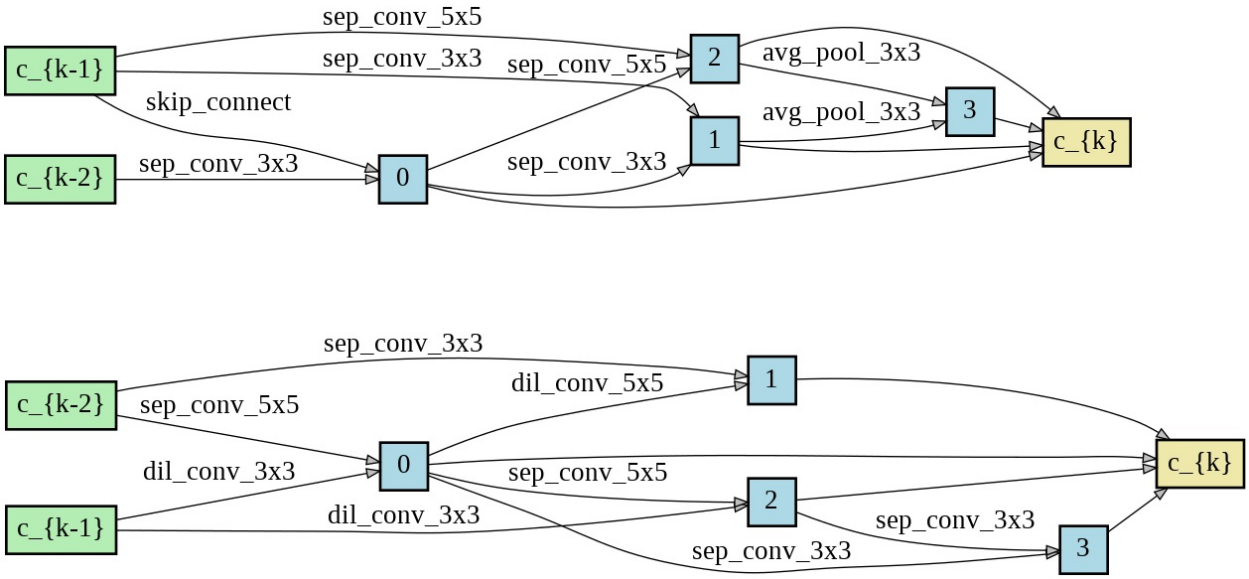

Figure 3.1: The architecture of the normal (top) and reduction (bottom) cells found by RARTS. This architecture contains only one skip connection. The last four edges are simply concatenated together to construct the next cell. So there is no search along these edges, following the convention of DARTS [37].

This discovered architecture is then transferred to and evaluated on ImageNet-1000, as shown in Table 3.2. The model found by RARTS of 4.7M parameters with 25.9% validation error is better than the one found by DARTS of 4.7M parameters with 26.7& error, and is also comparable to other differentiable methods. We may notice that ProxylessNAS, FairDARTS and PC-DARTS search the architecture directly on ImageNet-1000, instead of transferring the architecture learned on CIFAR-100. This can lead to higher accuracy, but may cost more computational resources. Many methods have evaluated the models via Tesla V100 GPUs, which have larger memory than the GTX 1080 Ti GPU used by DARTS and RARTS. For ImageNet-1000, the models are usually trained better by a GPU of larger memory as a larger batch size can be allowed. This can also partly explain the high accuracy of some methods on ImageNet, but a relatively low accuracy on CIFAR-100.

Finally, Table 3.3 shows the results of RARTS searching on NATS-Bench, which include the accuracy on the test data of CIFAR-10 and CIFAR-100, and the ratio of skip-connections in the discovered architectures. RARTS has surpassed both first-order DARTS and second-

Table 3.1: Comparison of DARTS, RARTS and other methods on CIFAR-10 based architecture search. DARTS-1/2 stands for DARTS 1st/2nd order, SNAS-Mi/Mo stands for SNAS plus mild/moderate constraints. Note that faster search times also depend on speed and memory capacity of local machines used. The V100 column indicates whether the model is trained on high-end Tesla V100 GPUs or not. The architecture discovered by RARTS is retrained and evaluated using a single GTX 1080 Ti GPU in our experiments. The numbers in the parentheses indicate the search GPU days of DARTS on our machine. Average of 5 runs. ⋄ These runs are conducted on our machine.

Method	Test Error (%)	Para. (M)	V100	Search GPU Days
Random Baseline [37]	3.29 ± 0.15	3.2	✗	4
AmoebaNet-B [43]	2.55 ± 0.05	2.8	✗	3150
SNAS-Mi [56]	2.98	2.9	✗	1.5
SNAS-Mo [56]	2.85 ± 0.02	2.8	✗	1.5
DARTS-1 [37]	3.00 ± 0.14	3.3	✗	1.5 (0.7)
DARTS-2 [37]	2.76 ± 0.09	3.3	✗	4 (3.1)
GDAS [19]	2.82	2.5	✓	0.2
ProxylessNAS [5]	2.08	5.7	✓	4.0
FairDARTS [11]	2.54 ± 0.05	3.3	✓	0.4
FairDARTS [11]	2.94 ± 0.05 ⋄	3.2	✗	0.3
P-DARTS [10]	2.50	3.4	✓	0.3
PC-DARTS [57]	2.57 ± 0.07	3.6	✓	0.1
PC-DARTS [57]	2.71 ⋄	2.9	✗	0.1
MiLeNAS [24]	2.80 ± 0.04	2.9	✓	0.3
MiLeNAS [24]	2.51 ± 0.11	3.9	✓	0.3
RARTS	2.65 ± 0.07	3.2	✗	1.1

order DARTS in accuracy by more than 20% on CIFAR-10 and 6% on CIFAR-100. In addition to its success in accuracy, RARTS has also completely avoided the architecture collapse problem, as the architecture found by RARTS contains no skip-connections. On the contrary, both architectures found by first-order DARTS and second-order DARTS contain 100% and 38.9% skip-connections on CIFAR-10 and CIFAR-100, respectively. This has also explained why the DARTS architectures are much worse in accuracy, and justified the advantage of RARTS from the architecture collapse viewpoint.

Table 3.2: Transfer to ImageNet: validation error comparison of DARTS, RARTS and other methods on local machines resp. The V100 column indicates whether the model is trained on high-end Tesla V100 GPUs or not. The larger GPU memory can support larger batch size, which leads to better accuracy and training efficiency on ImageNet. The Direct column indicates if the model is searched directly on ImageNet without transfer-learning. The direct search tends to be more accurate but costs more computational resources.

Method	Top-1 (%)	Top-5 (%)	Parameters (M)	V100	Direct
SNAS [56]	27.3	9.2	4.3	✗	✗
DARTS [37]	26.7	8.7	4.7	✗	✗
GDAS [19]	26.0	8.5	5.3	✓	✗
ProxylessNAS [5]	24.9	7.5	7.1	✓	✓
FairDARTS [11]	24.9	7.5	4.8	✓	✗
FairDARTS [11]	24.4	7.4	4.3	✓	✓
P-DARTS [10]	24.4	7.4	4.9	✓	✗
PC-DARTS [57]	25.1	7.8	5.3	✓	✗
PC-DARTS [57]	24.2	7.3	5.3	✓	✓
MiLeNAS [24]	25.4	7.9	4.9	✓	✗
RARTS	25.9	8.3	4.7	✗	✗

Table 3.3: Test errors of DARTS vs. RARTS on NATS-Bench search space. The results of DARTS on NATS-Bench are from [17]. Ratio = the number of skip-connections over the number of total operations in the discovered architecture.

Dataset	Method	Error (%)	Ratio (%)
CIFAR-10	DARTS-1	40.16	100
	DARTS-2	34.62	100
	RARTS	**11.48**	**0**
CIFAR-100	DARTS-1	38.74	38.9
	DARTS-2	39.51	38.9
	RARTS	**32.37**	**0**

Chapter 4

Compression of Neural Networks via Width Search

In this section, we consider the topic of network compression, its relation with width search and the application of the Relaxed Architecture Search method (RARTS) to network compression. First, we briefly go over the channel pruning methods for CNN layers, and formulate it in the way of width search by introducing the channel scoring parameters. Next, we propose a channel pruning and width search algorithm based on RARTS. In addition to the compression of the traditional CNN's, we also set up a framework for the compression of recent vision transformers from the viewpoint of architecture search. Finally, experiments are carried out on the common datasets for image classification and object detection.

4.1 Problem Formulation

4.1.1 Sparsification and Channel Pruning

Sparsification can help to construct efficient neural networks by removing redundant parameters from the over-parameterized networks, as the sparse networks we obtain occupy less memory and require fewer computational resources. It is common to add the ℓ_1 penalty [51] of all the model parameters to the loss and learn a sparse model during training. Different from the parameter shrinkage caused by ℓ_1, a regularization method based on ℓ_0 penalty has been proposed [40], resulting in a sparse network with reduced number of parameters and computations, less inference time and overfitting mitigated. Whereas their method has utilized stochastic binary gates and the hard-sigmoid to deal with the nonsmoothness of ℓ_0, the RVSM we have mentioned in equation (2.6) can solve the ℓ_0 regularized problem via an alternating gradient descent and closed-form update, generating sparse networks for classification tasks on CIFAR-10 [16] and medical images [58].

Compared with a sparse network of randomly distributed parameters of zero values, it is easier to realize the hardware efficiency by considering *group sparsity*, i.e., setting structured groups of weight parameters to be zero. A group sparsity method [55] has been developed by adding the group Lasso regularization [63], that is ℓ_1 norm on the ℓ_2 norm of each parameter group, to the loss function during training. The parameter group can be a channel in a convolutional layer, or the whole layer if the depth of the neural network needs to be reduced. The channel sparsity can immediately lead to *channel pruning*, which is a common method for generating efficient neural network through removing unimportant channels from a redundant network, where a channel is unimportant if its group ℓ_1 norm is small at the end of training. The loss function regularized by group ℓ_1 norm can be also optimized approximately using the group RVSM, which has been proved to be effective on multiple image classification tasks [59, 15, 4]. We shall be aware that when pruning the number

of channels by group sparsity, the regularization function can be written as $\mathcal{P}(w)$, which depends only on the weight parameters in the convolutional kernels.

4.1.2 Channel Pruning as Architecture Search

In addition to the conventional methods adding group sparsity to the convolutional weights, the channel pruning problem can be also formulated as a neural architecture search problem. A method called Network Slimming (NetSlim) has been proposed based on learning the channel scaling factors [38], which is able to reduce the model complexity and computational cost, and preserve the accuracy at the same time. These channel scaling factors are simply defined to be the learnable scale parameters γ of the batch normalization layer found in equation (3.3), and the channels corresponding to low scales are pruned. To learn sparse scales, the ℓ_1 regularization of these scale parameters is added to the loss during training. After being trained with ℓ_1 sparsity and the channels with low scales pruned, the model is further fine-tuned to achieve better performance. We shall be aware that the regularization term is not added to the convolutional weights, but directly to the scale parameters, which play a similar role as the architecture parameters in the differentiable neural architecture search context. This procedure of training a supernet, selecting a subnet, and training the selected subnet again is the same as that of DNAS. It is searching for the width indeed, whereas the DNAS methods mentioned in the previous section are searching for the topology. What makes NetSlim different from DNAS is its training algorithm, as it trains the scale parameters jointly with the convolutional weights without any splitting or alternating. As we have mentioned in the previous section, this may lead to overfitting.

Apart from NetSlim, there are many other channel pruning formulations and searching algorithms based on DNAS. TAS can search for both the width and depth of a network through sampling a few candidate feature maps with different number of channels [18]. These fea-

ture maps are aggregated by channel wise interpolation, and each of them is assigned a probability, parameterized by an architecture parameter which is learnable by DNAS. With constraints on FLOPS, a loss function is set up and optimized along with the probabilities via gradient descent. The structures with the highest probabilities are retained and the others are pruned. Following the steps of searching and pruning like the other methods, TAS has further used *knowledge distillation* [28] in the fine-tuning stage to improve the performance of the pruned network, which we will discuss in the next subsection. Despite the high accuracy of the model pruned by TAS on many classification tasks compared to other methods, it still remains unknown if the improvement of the accuracy comes from the proposed search algorithm or the fine-tuning stage with knowledge distillation. Moreover, the possible numbers of the channels can be searched are predefined, i.e., there are only 8 possible pruning ratios for each channel, ranging from 0%, 10% to 70%. Another search method called FasterSeg [9] is proposed for semantic segmentation tasks. Taking multi-resolution architectures into consideration, it can search for the width as well as operations and global topology like paths or the downsampling nodes for different resolutions. Similarly, it has also utilized knowledge distillation after the search stage and constrained the efficiency by a latency penalty. It has also applied DARTS for searching, and limited the pruning ratios to a predefined set of numbers like TAS.

Based on these previous works, we can summarize a general framework for searching the width of operations:

- Specify the architecture parameters for representing the width of the operations
- Set up a loss function which involves the architecture parameters and the other learnable parameters, with efficiency constraints on the architecture parameters
- Optimize the loss via gradient descent or DNAS and prune the network based on the values of the architecture parameters

- Fine-tune the pruned network, possibly with knowledge distillation

This guideline can be adjusted accordingly for different networks and tasks. We will see in the next sections how we complete each step and find the desired width with the assistance of RARTS.

4.2 Search for the Width of CNNs

4.2.1 Method

First, we need to determine a way to specify the architecture parameters. From the discussion in the above section, NetSlim seems to be more flexible since each convolutional layer can preserve any number of channels after pruning, whereas the other methods can retain only a few different pruning ratios. However, NetSlim sets the channel scoring parameters to be the scales of the batch normalization layers, which are not reliable when the batch size is too small. A network without any batch normalization layers is common when the task is complicated and the GPU memory is limited [20, 39]. Hence, we adopt the following simple but universal method to determine the architecture parameters α by assigning them directly to the feature maps through pointwise multiplication:

$$\widetilde{\boldsymbol{X}}^i = \alpha^i \boldsymbol{X}^i, \tag{4.1}$$

where $\boldsymbol{X}^i \in \mathbb{R}^{N \times 1 \times H \times W}$ and $\alpha^i \in \mathbb{R}$ are the output feature of the i-th convolutional channel and its corresponding architecture parameter, $\widetilde{\boldsymbol{X}}^i$ is the input feature of the i-th channel for the next layer. Although equation (4.1) does not contain any convolution operations explicitly, one shall be aware that multiplying the i-th output feature map by α^i is equivalent to multiplying the i-th convolutional filter which produces this output feature map by the

same α^i. Therefore, we can prune the convolutional channels whose output features have small values of α.

Next, we need to set up a loss function including both the weight parameters and the architecture parameters. Since we will apply RARTS to optimize the loss, we can directly use the Lagrangian as defined in equation (2.11) with a penalty on the architecture parameters like that the ℓ_1 in NetSlim. We obtain the search, pruning and fine-tuning algorithm:

Algorithm 3: Search for the width via RARTS

Input: the number of iterations N, the hyperparameters λ, β and γ, a learning rate schedule $(\eta_w^t, \eta_u^t, \eta_\alpha^t)$, a pruning ratio m, initialization of the weight parameters w^0, u^0 and the architecture parameters α^0.

Output: a pruned model with the fine-tuned weight parameters w

The search stage: Split the dataset $\mathcal{D}$ into two subsets $\mathcal{D}_p$ and $\mathcal{D}_a$.

for $t = 0, 1, ..., N$ **do**

Compute $\mathcal{L}_p$ and $\mathcal{L}_a$ on $\mathcal{D}_p$ and $\mathcal{D}_a$, respectively, and then compute $\mathcal{L}$ using equation (2.11)

Update the parameters via gradient descent:

$w^{t+1} = w^t - \eta_w^t \nabla_w \mathcal{L}(u^t, w^t, \alpha^t)$

$u^{t+1} = u^t - \eta_u^t \nabla_u \mathcal{L}(u^t, w^{t+1}, \alpha^t)$

$\alpha^{t+1} = \alpha^t - \eta_\alpha^t \nabla_\alpha \mathcal{L}(u^{t+1}, w^{t+1}, \alpha^t)$

end

The pruning stage: Sort the α and prune the corresponding channels according to the ratio m.

The fine-tuning stage: Fine-tune the pruned model with a warm start or knowledge distillation if a pretrained full-size model is given. We use the regular gradient descent or SGD only to learn the weight parameters w in this stage.

For the knowledge distillation part, we add an extra loss $\mathcal{L}_{KD}$ to the overall loss for fine-

tuning the model, following that in TAS [18]:

$$\mathcal{L}_{KD} = \frac{1}{N} \sum_{n} \sum_{i} \sigma_i(\hat{\boldsymbol{U}}^n/T) \log \big(\sigma_i((\hat{\boldsymbol{U}}^n/T)/\sigma_i((\boldsymbol{U}^n/T)\big),$$

where $\hat{\boldsymbol{U}}^n$ and $\boldsymbol{U}^n$ are the output logits of the unpruned and pruned networks in the n-th batch, σ_i is the i-th softmax function, and T is a hyperparameter. We shall note that the loss is equal to the KL divergence (equation (1.3)) of the pruned model's predicted probability distribution from the unpruned model's probability distribution. In other words, this extra loss helps the pruned model to learn the soft labels of the unpruned model.

4.2.2 Experiments

We search for the width of PreResNet-164 [27] on CIFAR-10 and CIFAR-100. PreResNet is slightly different from the original ResNet [26] in that it has regarded the batch normalization layer and the ReLU function as the pre-activation, i.e., the activation before the convolutional layer. The pre-activation structure is claimed to have regularization effects, make the optimization easier, and performs better than the original ResNet on CIFAR datasets. Each residual block of PreResNet-164 contains three sequential modules, with each of the module contains a batch normalization, a ReLU, and a convolutional layer sequentially. It also follows the *bottleneck* design of the original ResNet to reduce the computational costs. That is to say, the first and the last convolutions have 1×1 kernels for reducing and expanding the number of channels, whereas the intermediate convolutions have 3×3 kernels for learning the features. Suppose the width reduction ratio is $1/4$, we have an input feature map $\boldsymbol{X} \in \mathbb{R}^{1\times 4C\times H\times W}$ of batch size one, and the three convolutions can be represented by $\boldsymbol{K}_1 \in \mathbb{R}^{C\times 4C\times 1\times 1}$, $\boldsymbol{K}_2 \in \mathbb{R}^{C\times C\times k\times k}$, and $\boldsymbol{K}_3 \in \mathbb{R}^{4C\times C\times 1\times 1}$. Then this bottleneck residual block has a parameter number of $O(C^2k^2 + 8C^2)$, and a computational complexity of $O(C^2k^2HW + 8C^2HW)$. Suppose the remaining ratio of the channels is a constant ρ for the

three convolutions. Then the pruned convolutions are $\boldsymbol{K}'_1 \in \mathbb{R}^{C\times 4\rho C\times 1\times 1}$, $\boldsymbol{K}'_2 \in \mathbb{R}^{\rho C\times \rho C\times k\times k}$, and $\boldsymbol{K}'_3 \in \mathbb{R}^{4C\times \rho C\times 1\times 1}$. The resulting number of parameter is $O(C^2\rho^2k^2 + 8\rho C^2)$, and the computational complexity is $O(C^2\rho^2k^2HW + 8\rho C^2HW)$.

When using RARTS to search for width, we follow the hyperparameters and settings of NetSlim as well. That is, learning rate = 0.1, weight decay = 0.0001, epochs = 160, and the sparsity scale $\gamma = 0.0001$ [38]. The same settings also apply when fine-tuning the pruned model, except for the distillation parameter $T = 4$, following that of TAS [18]. We present the results of the pruned model without and with distillation. In Table 4.1, RARTS without distillation outperforms NetSlim (NS) and TAS [18] by around 10% error reduction on CIFAR-10. While TAS does not offer an option to specify the pruning ratio of channels (PRC), the pruning ratio of FLOPs is around 30% for NS (40% PRC), RARTS (40% PRC) and TAS. So the comparison is fair. On CIFAR-100, RARTS still leads NetSlim at the same PRC. The gap is smaller as the baseline network is less redundant. Our experimental results reveal that the accuracy of TAS with knowledge distillation is lower than (on CIFAR-10) or similar to (on CIFAR-100) that of RARTS, while TAS without distillation is 2% worse [18]. Actually, the column with distillation (KD) shows that the performance of RARTS pruning can be further improved, especially on a more complicated dataset like CIFAR-100. These comparisons support the fact that RARTS works better as a differentiable method for width search, without regard to any other training tricks.

Apart from the comparisons with the above methods, we also consider a pruning task for comparing DARTS and RARTS, which can be viewed as an ablation study of RARTS on the width search task. For this task, we prune MobileNetV2 [46] on a randomly sampled 20-class subset of ImageNet-1000, with ℓ_1 regularization but unfixed pruning ratio. This subset (which is denoted by ImageNet-20) contains 26,000 images for training and 1,000 images for validation, which is much fewer than ImageNet-1000. The pruning ratio can be learned automatically by the strong regularization term, as many of the architecture parameters are

Table 4.1: Application of RARTS to PreResNet-164 (baseline, 1.7 M parameters) channel pruning on CIFAR-10 and CIFAR-100, in comparison with the baseline, TAS and NetSlim. The numbers in the parentheses indicate the pruning ratio of channels (PRC). For NetSlim and RARTS, PRC is fixed at 40% or 60%. KD means the test error (%) with the use of knowledge distilled from the unpruned model. NS = NetSlim.

Data	Method	Test Error (%)	KD	FLOPS
CIFAR-10	Baseline [38]	4.22	-	2.48×10^8
	TAS [18]	-	6.00	1.78×10^8
	NS (40% PRC) [38]	5.08	-	1.90×10^8
	RARTS (40% PRC)	**4.58**	**4.58**	1.90×10^8
	NS (60% PRC) [38]	5.27	-	1.38×10^8
	RARTS (60% PRC)	**4.90**	5.01	1.33×10^8
CIFAR-100	Baseline [38]	21.83	-	2.48×10^8
	TAS [18]	-	22.24	1.71×10^8
	NS (40% PRC) [38]	22.87	-	1.67×10^8
	RARTS (40% PRC)	22.64	**21.63**	1.78×10^8
	NS (60% PRC) [38]	23.91	-	1.24×10^8
	RARTS (60% PRC)	23.26	**22.38**	1.23×10^8

simply zero. Table 4.2 shows that RARTS also beats both random pruning and DARTS in accuracy. Even though the 2nd DARTS obtains a higher sparsity, it sacrifices the accuracy.

4.3 Search for the Dimensions of Transformers

4.3.1 Background

Unlike the convolutional or recurrent neural networks (CNN or RNN) [23], *transformers* are the models based completely or partially on the *attention* mechanisms. They are originally proposed to learn global dependency for sequence transduction tasks [53], , and have obtained better performance and training efficiency. The general architecture of the transformer for sequence modeling is composed of an encoder module and a subsequent decoder module. The encoder module is a stack of a few sequential encoder blocks, with each of them containing

Table 4.2: Application of RARTS to MobileNetV2 pruning on the ImageNet-20 dataset (a randomly sampled subset of ImageNet-1000, with 20 object classes), compared with the baseline, random pruning, 1st and 2nd order DARTS. Here random pruning means that we zero out channels randomly in accordance with the pruning ratio of RARTS. Average of 5 runs. PRC = the average pruning ratio of channels over the pruned layers. We note that the PRC can be high because the dataset is much smaller.

Method	Test Error. (%)	PRC (%)
Baseline	12.3 ± 1.4	-
Random Pruning	12.0 ± 1.1	71.2 ± 1.9
DARTS-1	10.1 ± 2.0	69.0 ± 0.9
DARTS-2	9.8 ± 1.7	72.6 ± 2.0
RARTS	**8.2 ± 1.9**	71.2 ± 1.9

a *self-attention* (SA) layer and a fully connected *feed-forward network*. The encoder block has also adopted the residual structure [26], with a layernorm applied after the summation of the shortcut and the residual.

While the feed-forward network consists simply of two fully connected layers, the self-attention layer is computed through a *multi-head attention* (MSA) mechanism, which is more complicated and usually requires more computational resources than the convolution operations used in CNNs. Specifically, the input features are first embedded to a triple of *queries*, *keys* and *values*, and then distributed to a few heads. For each head, the attention map is computed via the scaled dot-product of the queries and the keys, and then assigned to the values. After that, the updated values from each head are concatenated and projected to construct the output features. This mechanism is claimed to perform better than the single-head attention as different heads are believed to learn different representation subspaces [53]. The decoder module is also a stack of a few sequential decoder blocks. However, each decoder block further contains a multi-head attention layer, in addition to the multi-head self-attention and the feed-forward network of the encoder. This extra multi-head attention is computed with the keys and values from the encoder module and the queries from the decoder.

Besides its success in language models, transformers have also been widely studied in computer vision tasks. One of the directions is to replace the CNN backbones by transformers. In other words, transformers are used to extract features from images, and the features are processed by various heads to solve various tasks after that. Vision Transformer (ViT) [20] is among the vision models whose backbones are purely transformers. ViT has partitioned the input image into small patches to mimic the tokens in the language transformers. Instead of pixels, these patches are embedded into features of certain dimensions, serving as the input of the attention module. Since its job is to learn representations, ViT has included the encoder module only, i.e., a stack of multi-head self-attentions. In spite of ViT's high accuracy on image classification, there are some concerns about its quadratic computational complexity on the number of queries n. That means the complexity is also quadratic on the input resolution $H \times W$, whereas the convolution operation has linear complexity. ViT has also been restricted to classification, since pixel-level tasks like segmentation typically need to deal with high resolution features.

A window-based transformer called Swin Transformer [39] has then been proposed for these more complicated vision tasks. Similar to ViT, Swin has also provided a series of backbones which are based purely on transformers, especially the transformer encoders. The first advantage of Swin is that it can generate hierarchical features so that they can be used to solve semantic segmentation and object detection tasks with suitable heads. To obtain features of different resolutions, Swin has merged $2 \times 2 = 4$ image patches into 1 patch at the end of each architecture stage. Since the size of patches is fixed, the image height and the width are both reduced by a half after merging. The overall transformer architecture is divided into one initial stage without merging and three intermediate stages with merging, and hence it can produce features of four resolution levels. Another advantage comes from the window-based multi-head self-attention (W-MSA) with shifting. Compared with the quadratic complexity of MSA, W-MSA has achieved a linear complexity from computing the attentions locally, within a small window of patches. Global information across different

windows is then exchanged via shifting the window partitions.

Similar to the channel pruning in CNNs, there are also some studies for vision transformer pruning. Inspired by NetSlim, VTP [67] has assigned scoring parameters to the features before the linear embedding or projection layers and pruned the dimensions of these features which are corresponding to low scores. Since the dimensions of the linear layers depend on the dimensions of the input features, the parameters of these layers are also reduced. Another pruning method has been proposed in NViT [60], which is based on the scores of grouped structural parameters. The scores are different from those of VTP as they are computed directly from the weight parameters. NViT has taken pruning the number of heads and the latency on hardware into account. Moreover, it has been pointed out that having the same dimensions across all layers in the conventional transformer design might not be optimal [60], which encourages the studies of automated transformer architecture design.

These pruning methods have obtained high pruning ratio with a very small accuracy loss for vision transformers like DeiT [52], on the image classification tasks. It would be natural to consider pruning Swin or other light transformer backbones for multiple computer vision tasks. WDPruning [61] is a direct pruning method for Swin on ImageNet classification, without the fine-tuning stage. It has also provided an option for depth pruning, and an automated learned pruning ratio based on learnable thresholds of saliency scores. However, experimental results has shown worse accuracy of the pruned models, as it has not been fine-tuned. Therefore, we propose in this section a pruning method for transformer backbone which is valid on both image classification and object detection tasks. Since our method aims to search for the intrinsic dimensions (i.e., the possible lowest dimensions to maintain network performance) of transformers, we name it SiDT in the rest of this section. Although SiDT is inspired by previous pruning methods like Network Slimming [38] and Vision Transformer Pruning (VTP) [67], it has its own merits:

- SiDT can prune transformers for not only classification tasks, but also other vision tasks like object detection.
- We have analyzed the computational complexity of the unpruned and the pruned models.
- The models with 20% or 40% dimensions pruned perform similarly or even better than the unpruned model.
- SiDT prunes the dimensions of linear embeddings, different from the feature pruning of VTP.

4.3.2 Method

Architecture parameters. For the dimension search of transformers, we still follow the four stages summarized at the end of Section 4.1. Since the searching, pruning and fine-tuning stages are similar, the key difference is how we set up the architecture parameters. Whereas we prune convolution operations in CNNs, there are a few types of operations for different transformers. So we discuss in detail the strategies of setting up architectures parameters for MSA, W-MSA and *multilayer perceptron* (MLP) [39]. Suppose again the batch size is $N = 1$, $\boldsymbol{X} \in \mathbb{R}^{1\times d\times H\times W}$ is the input feature map with H and W the resolution and d the dimension of the feature. Set $n = H \times W$, we obtain the transformed input feature $\boldsymbol{X} \in \mathbb{R}^{n\times d}$.

For SA, $\boldsymbol{X}$ is linearly embedded into the query $\boldsymbol{Q}$, key $\boldsymbol{K}$ and value $\boldsymbol{V}$ of the same shapes:

$$\boldsymbol{Q} = \boldsymbol{X}\,\boldsymbol{W}_Q,\ \boldsymbol{K} = \boldsymbol{X}\,\boldsymbol{W}_K,\ \boldsymbol{V} = \boldsymbol{X}\,\boldsymbol{W}_V,$$

where the embedding matrices $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, $\boldsymbol{W}_V \in \mathbb{R}^{d\times d}$, if the embedding dimensions for the query, key and value are also equal to d. Then the attention map a is computed via the

softmax function σ of the scaled product of the query and the key::

$$a(\boldsymbol{Q}, \boldsymbol{K}) = \sigma(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d}) \in \mathbb{R}^{n\times n},$$

and assigned to the value to compute the output of SA:

$$SA(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \sigma(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d})\,\boldsymbol{V} \in \mathbb{R}^{n\times d}.$$

Note that the output of SA has the same shape as the input $\boldsymbol{X}$. To set up the architecture parameters, we apply a uniform *scoring matrix* $\boldsymbol{A}$ for $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ via matrix multiplication:

$$\widetilde{\boldsymbol{Q}} = \boldsymbol{Q}\boldsymbol{A},\ \widetilde{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{A},\ \widetilde{\boldsymbol{V}} = \boldsymbol{V}\boldsymbol{A},$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is a diagonal matrix whose diagonal elements are the architecture parameters α_i. That is to say, we assign a score α_i to the i-th dimension of the d-dimensional query, and also to the key and value at the same i-th dimension. Then we compute the SA module based on the scored query, key and value, and obtain $SA(\widetilde{\boldsymbol{Q}}, \widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}})$.

For MSA, we need to compute multiple SA modules and each of them is a *head*. Let h be the number of heads. For $j = 1, ..., h$, we also compute $\boldsymbol{Q}_j$, $\boldsymbol{K}_j$ and $\boldsymbol{V}_j \in \mathbb{R}^{n\times d/h}$ through linear embedding of $\boldsymbol{X}$ via $\boldsymbol{W}_{Q,j}$, $\boldsymbol{W}_{K,j}$ and $\boldsymbol{W}_{V,j} \in \mathbb{R}^{d\times d/h}$ like that of SA, and obtain the heads:

$$\boldsymbol{H}_j = SA(\boldsymbol{Q}_j, \boldsymbol{K}_j, \boldsymbol{V}_j) \in \mathbb{R}^{n\times d/h}.$$

With $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ the concatenations of $\boldsymbol{Q}_j$, $\boldsymbol{K}_j$ and $\boldsymbol{V}_j$, the output of the MSA module is computed by concatenating the heads and projecting linearly via $\boldsymbol{W}_O \in \mathbb{R}^{d\times d}$:

$$MSA(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\boldsymbol{H}_1, \boldsymbol{H}_2, ..., \boldsymbol{H}_h]\,\boldsymbol{W}_O \in \mathbb{R}^{n\times d}.$$

We use a stronger scoring matrix $\boldsymbol{A} \in \mathbb{R}^{d/h \times d/h}$ for MSA, which is not only uniform over the query, key and value, but also over all the heads:

$$\widetilde{\boldsymbol{Q}}_j = \boldsymbol{Q}_j \boldsymbol{A},\ \widetilde{\boldsymbol{K}}_j = \boldsymbol{K}_j \boldsymbol{A},\ \widetilde{\boldsymbol{V}}_j = \boldsymbol{V}_j \boldsymbol{A},$$

for $j = 1, 2, ..., h$. Then we compute the new MSA module and obtain $\widetilde{\boldsymbol{H}}_j = SA(\widetilde{\boldsymbol{Q}}_j, \widetilde{\boldsymbol{K}}_j, \widetilde{\boldsymbol{V}}_j)$ and:

$$MSA(\widetilde{\boldsymbol{Q}}, \widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}) = [\widetilde{\boldsymbol{H}}_1, \widetilde{\boldsymbol{H}}_2, ..., \widetilde{\boldsymbol{H}}_h]\, \boldsymbol{W}_O.$$

For W-MSA, the input features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ are divided into a few windows of size $M \times M$, and MSA is computed locally within these windows. That is to say, we reshape $\boldsymbol{X}$ to be a tensor in $\mathbb{R}^{n/M^2 \times M^2 \times d}$, and obtain $\boldsymbol{Q}_j$, $\boldsymbol{K}_j$ and $\boldsymbol{V}_j \in \mathbb{R}^{n/M^2 \times M^2 \times d/h}$ for $j = 1, 2, ..., h$ after embedding of multi-head. Here $\boldsymbol{Q}_j$, $\boldsymbol{K}_j$ and $\boldsymbol{V}_j$ can be viewed as the concatenations of $\boldsymbol{Q}_{j,l}$, $\boldsymbol{K}_{j,l}$ and $\boldsymbol{V}_{j,l} \in \mathbb{R}^{M^2 \times d/h}$ for $l = 1, 2, ..., n/M^2$. For each window, we compute the MSA module and obtain $\boldsymbol{W}_{,l} = MSA(\boldsymbol{Q}_{,l}, \boldsymbol{K}_{,l}, \boldsymbol{V}_{,l}) \in \mathbb{R}^{M^2 \times d}$. Finally, we rearrange the outputs of these windows and obtain:

$$W\text{-}MSA(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\boldsymbol{W}_{,1}, \boldsymbol{W}_{,2}, ..., \boldsymbol{W}_{,n/M^2}] \in \mathbb{R}^{n \times d}.$$

To set up the architecture parameters for W-MSA, again we use a uniform scoring matrix $\boldsymbol{A} \in \mathbb{R}^{d/h \times d/h}$ for the query, key and value, over all the heads and windows:

$$\widetilde{\boldsymbol{Q}}_{j,l} = \boldsymbol{Q}_{j,l} \boldsymbol{A},\ \widetilde{\boldsymbol{K}}_{j,l} = \boldsymbol{K}_{j,l} \boldsymbol{A},\ \widetilde{\boldsymbol{V}}_{j,l} = \boldsymbol{V}_{j,l} \boldsymbol{A}.$$

Then we have $\widetilde{\boldsymbol{W}}_{,l} = MSA(\widetilde{\boldsymbol{Q}}_{,l}, \widetilde{\boldsymbol{K}}_{,l}, \widetilde{\boldsymbol{V}}_{,l})$ and

$$W\text{-}MSA(\widetilde{\boldsymbol{Q}}, \widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}) = [\widetilde{\boldsymbol{W}}_{,1}, \widetilde{\boldsymbol{W}}_{,2}, ..., \widetilde{\boldsymbol{W}}_{,n/M^2}].$$

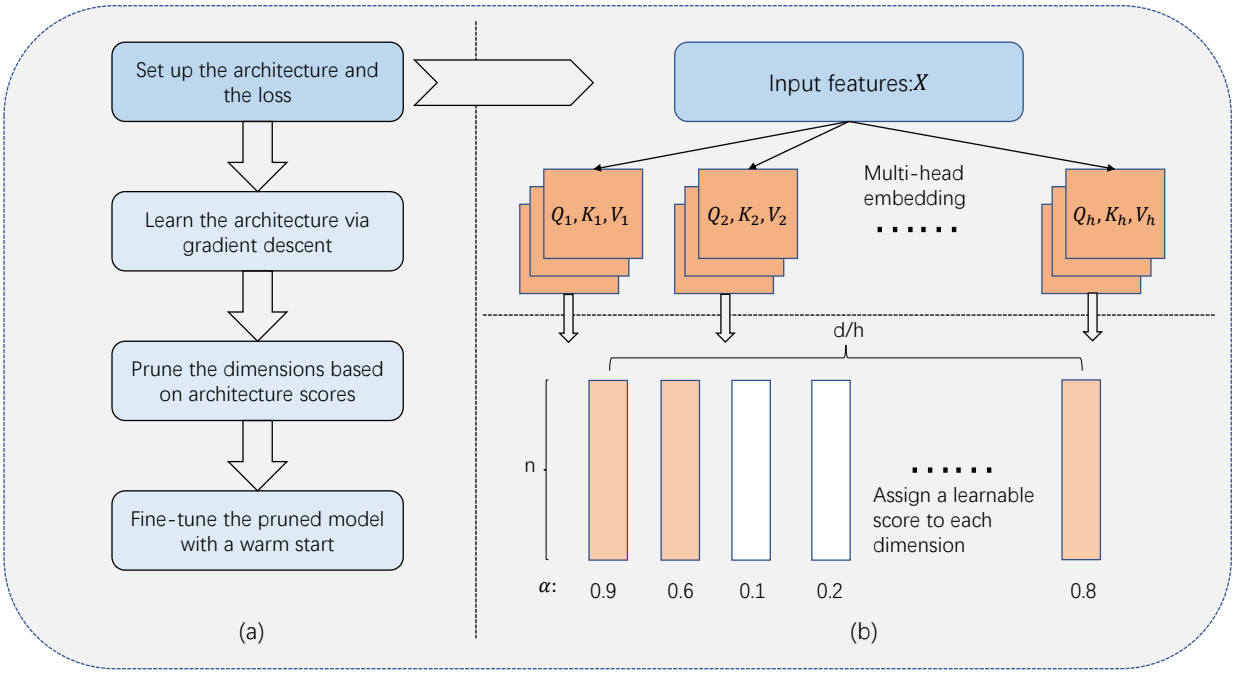

Figure 4.1: (a) The stages of transformer pruning. (b) Assign the scoring matrix $\boldsymbol{A} = \text{diag}(\alpha)$ to the output dimensions of multi-head queries, keys and values.

The last module to be discussed is MLP [39], which simply contains two linear layers with activation. Suppose $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the input feature, and d_m represents the dimensions of the hidden state. Suppose further $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$ are two matrices for linear embedding, σ_{MLP} is the activation. Then we have:

$$MLP(\boldsymbol{X}) = \sigma_{MLP}(\boldsymbol{X}\,\boldsymbol{W}_1)\,\boldsymbol{W}_2 \in \mathbb{R}^{n \times d}.$$

The scoring matrix $\boldsymbol{A}$ is applied immediately after $\boldsymbol{W}_1$ through matrix multiplication, and get $\sigma_{MLP}(\boldsymbol{X}\,\boldsymbol{W}_1\boldsymbol{A})\,\boldsymbol{W}_2$. Here $\boldsymbol{A}$ can be viewed as the scores for the dimensions of the hidden state.

Pruning. The four-stage pruning procedure is summarized in Section 4.1.2 and also in Fig. 4.1. During the searching stage, the elements in the scoring matrix $\boldsymbol{A}$ are updated via gradient descent or DNAS algorithms like RARTS, together with the elements of the

embedding matrices $\boldsymbol{W}$. After the completion of searching, we rank the diagonal elements of the scoring matrix $\boldsymbol{A}$ according to their absolute values. The dimensions of the embedding matrices are pruned if their corresponding scores are ranked low. Suppose the remaining ratio of the dimensions after pruning is ρ. Then only ρd dimensions with higher scores are left in the pruned matrices.

For MSA, we have $\boldsymbol{W}_{Q,j}$, $\boldsymbol{W}_{K,j}$ and $\boldsymbol{W}_{V,j} \in \mathbb{R}^{d\times\rho d/h}$ after pruning, and hence $\boldsymbol{Q}_j$, $\boldsymbol{K}_j$ and $\boldsymbol{V}_j \in \mathbb{R}^{n\times\rho d/h}$. Since we have not pruned the query or key number n, the attention map still belongs to $\mathbb{R}^{n\times n}$, and the head $\boldsymbol{H}_j \in \mathbb{R}^{n\times\rho d/h}$. This leads to the projection matrix $\boldsymbol{W}_O \in \mathbb{R}^{\rho d\times d}$, and the output of the pruned MSA in $\mathbb{R}^{n\times d}$, with the same shape as the unpruned model. One can easily see that the original unpruned MSA module has $O(4d^2)$ parameters and a computational complexity of $O(4nd^2 + 2n^2d)$. For the pruned MSA, the number of parameters is reduced to $O(4\rho d^2)$, and the computational complexity is reduced to $O(4\rho nd^2 + 2\rho n^2 d)$. Similarly, the unpruned W-MSA module has $O(4d^2)$ parameters and a computational complexity of $O(4nd^2 + 2nM^2d)$. For the pruned W-MSA, the number of parameters is reduced to $O(4\rho d^2)$, and the computational complexity is reduced to $O(4\rho nd^2 + 2\rho nM^2d)$. Finally, the unpruned MLP has $O(2dd_m)$ parameters and a computational complexity of $O(2ndd_m)$. For the pruned MLP, the number of parameters is reduced to $O(2\rho dd_m)$, and the computational complexity is reduced to $O(2\rho ndd_m)$. This is because $\boldsymbol{W}_1 \in \mathbb{R}^{d\times\rho d_m}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{\rho d_m\times d}$ after pruning. The complexity of operations before and after pruning is summarized in Table 4.3.

One shall note that our settings of architecture parameters are different from those of VTP [67]. VTP's scoring matrix $\boldsymbol{A}$ is applied directly to the input feature $\boldsymbol{X}$, whereas ours is applied to $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$. In other words, VTP prunes the features but we prune the linear embeddings. As we apply the same matrix $\boldsymbol{A}$ to embedding dimensions of multiple heads, we have only d/h such architecture parameters, making the model easier to train. Moreover, VTP is applied to DeiT on the classification task only, whereas our method prunes Swin

Table 4.3: Complexity of different operations before and after pruning. We suppose the remaining dimension ratio ρ is fixed over all the operations, so that the complexity estimation is easier.

Operation	Pruned	Para.	Complexity
MSA	✗	$O(4d^2)$	$O(4nd^2 + 2n^2d)$
	✓	$O(4\rho d^2)$	$O(4\rho nd^2 + 2\rho n^2 d)$
W-MSA	✗	$O(4d^2)$	$O(4nd^2 + 2nM^2d)$
	✓	$O(4\rho d^2)$	$O(4\rho nd^2 + 2\rho nM^2d)$
MLP	✗	$O(2dd_m)$	$O(2ndd_m)$
	✓	$O(2\rho dd_m)$	$O(2\rho ndd_m)$

Transformer, which serves as a backbone for multiple vision tasks. Finally, we have also provided the complexity analysis of the unpruned and pruned operations, which is missing in previous studies.

4.3.3 Experiments

We first conduct SiDT for Swin Transformer on CIFAR-100 image classification. We prune its tiny version (Swin-T), which has 27.53M parameters and a complexity of 4.49G FLOPS. The settings of the search stage are similar to those for training the unpruned baseline[1], with batch size = 256, patch size = 4, window size = 7, embedding dimension = 96, initial learning rate = 0.00025, momentum =0.9, weight decay = 0.05, epochs = 160, and the sparsity scale $\gamma = 0.0001$ for ℓ_1 regularization. After searching, we obtain the scores of all the dimensions and rank them according to their absolute values. Next, the dimensions with lower scores are pruned, based on a predefined pruning ratio of 20%, 40%, 60% and 80%. Finally, the pruned model is trained again with a warm start, using the same settings as the search stage.

[1]When setting up the architecture parameters, we refer to the code at https://github.com/Cydia2018/ViT-cifar10-pruning

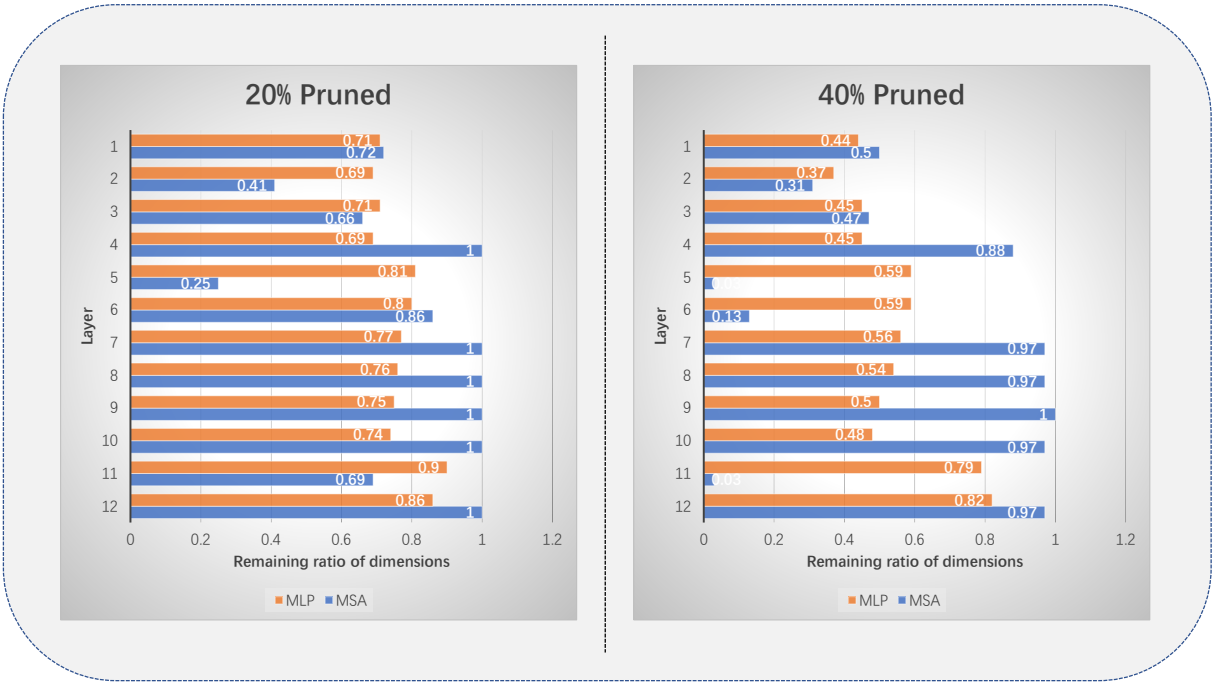

Figure 4.2: Ratio of the remaining dimensions over different layers in the networks with 20% or 40% dimensions pruned.

Table 4.4 shows that the number of parameters and computational costs can be greatly reduced after pruning, while preserving the accuracy at the same time, compared to the baseline [66]. The visualization of the remaining dimensions after pruning can be found in Fig. 4.2. After pruning 80% of the dimensions, the accuracy is only around 2% lower than the recovered baseline. The model with 20% or 40% dimensions pruned has an accuracy which is even higher than the baseline model. This can be explained by the relatively larger size of Swin-T on easier datasets like CIFAR, as over-parameterized models can cause overfitting. To verify this phenomenon, we have also transferred Swin-T and prune it again on the ImageNet-20 dataset, which is a 20-class subset of ImageNet-1000, as described in Section 4.2.2. As shown in Table 4.5, the model with 20% and 40% dimensions pruned still beat the unpruned baseline in accuracy. For the model with 80% dimensions pruned, the accuracy drop is less than 1%.

Additionally, we have also pruned the Swin-T backbone for the COCO object detection task

Table 4.4: Prune Swin-T via SiDT on CIFAR-100 classification task. PR = Pruning Ratio. Acc = accuracy. Para. = number of parameters. ⋄ The baseline is recovered on our device of one RTX 3090 GPU.

PR	Acc (%)	Para. (M)	FLOPS (G)
0% (Baseline [66])	78.07	-	-
0% (Baseline ⋄)	81.78	27.60	4.49
20%	**82.75**	23.28	3.53
40%	82.11	17.89	2.60
60%	80.81	11.92	1.73
80%	79.35	**7.17**	**0.92**

Table 4.5: Prune Swin-T via SiDT on ImageNet-20 classification task. PR = Pruning Ratio.

PR	Acc (%)	Para. (M)	FLOPS (G)
0% (Baseline)	91.7	27.53	4.49
20%	**92.5**	20.34	3.65
40%	91.7	14.61	2.82
60%	91.3	10.06	1.93
80%	90.9	**5.93**	**1.09**

[35], following the settings in the Swin paper [39]. That is, batch size = 16, initial learning rate = 0.0001, weight decay = 0.05, epochs = 36, and all the other settings of the backbone are the same as the Swin-T for CIFAR classification discussed above. We use Cascade Mask R-CNN [6] as the detection head, in accordance with that of the Swin-T baseline. Again we follow the steps in Fig. 4.1, and prune the model with pruning ratios of 20% and 40%. During the search stage, we also start training with a pretrained Swin-T object detection model. Table 4.6 indicates that the model with 20% dimensions of the backbone pruned has a similar performance of box mAP and mask mAP as the unpruned model. Even if 40% dimensions of the backbone are pruned, the loss in AP is still less than 1.5%. This is a fair result since the detection task is more complicated than the classification task, and pruning a detection model can lead to a slightly larger accuracy decline.

Table 4.6: Prune Swin-T backbone via SiDT on COCO object detection task. PR = Pruning Ratio. ⋄ This baseline is recovered on our device.

PR	mAP		Para. (M)	
	Box	Mask	Total	Backbone
0% (Baseline [39])	50.5	43.7	86	28
0% (Baseline ⋄)	50.6	43.9	86	28
20%	50.4	43.7	80	22
40%	49.2	42.9	74	16

Chapter 5

Conclusion

We have proposed RARTS, a first order efficient method for searching architectures of neural networks. The method is based on splitting both the data and the network, and the loss function with network parameters splitting is optimized via a three-step alternating gradient descent. We have proved a convergence theorem when the Lipschitz gradient assumption holds on the loss function. Further result of convergence rate is proved if there is an extra assumption of convexity on the loss. It has been revealed by an analytical example which satisfies these conditions that RARTS converges better than DARTS, the differentiable method developed by previous researchers.

We have conducted experiments on searching for the topological architectures of neural networks. The image classification results on CIFAR-10, ImageNet-1000 and NATS-Bench demonstrate the better performance of RARTS over DARTS. During the search stage, We have taken hardware latency into account, in the form of penalized loss. In addition, we have also applied it to the width search of the convolution operations, and pruned the CNN models based on the search results. Experiments of pruning PreResNet-164 on CIFAR classification datasets have supported its superiority over the counterparts as well. The models with 20%

or more parameters and computations pruned can perform similarly or even better than the unpruned model.

Finally, we have extended the proposed width search method to searching for the dimensions of vision transformers on multiple vision tasks. Results of pruning Swin-T on CIFAR-100 classification and COCO object detection tasks are aslo convincing, like those for pruning CNNs. There are still many topics to be done in the future work. As many algorithms are raised for topology search, we would like to transfer them to width search, in order to figure out a unified optimal method. Moreover, there are many manually designed models for other computer vision tasks like segmentation. We will investigate the applications of our methods in the automated learning of their architectures.

Bibliography

[1] T. Blumensath and M. E. Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[3] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] K. Bui, F. Park, S. Zhang, Y. Qi, and J. Xin. Structured sparsity of convolutional neural networks via nonconvex sparse group regularization. *Frontiers in applied mathematics and statistics*, 6, 2021.

[5] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2018.

[6] Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[9] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang. Fasterseg: Searching for faster real-time semantic segmentation. In *International Conference on Learning Representations*, 2019.

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

[11] X. Chu, T. Zhou, B. Zhang, and J. Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European conference on computer vision*, pages 465–480. Springer, 2020.

[12] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

[13] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

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

[15] T. Dinh, B. Wang, A. Bertozzi, S. Osher, and J. Xin. Sparsity meets robustness: channel pruning for the feynman-kac formalism principled robust deep neural nets. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 362–381. Springer, 2020.

[16] T. Dinh and J. Xin. Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1, ℓ_0, and transformed-ℓ_1 penalties. In *Proceedings of SAI Intelligent Systems Conference*, pages 360–374. Springer, 2020.

[17] X. Dong, L. Liu, K. Musial, and B. Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[18] X. Dong and Y. Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 760–771, 2019.

[19] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 1761–1770, 2019.

[20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[21] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[22] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd ed.)*. Johns Hopkins Univ. Press, 1996.

[23] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[28] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[30] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

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

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[35] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[36] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[37] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *ICLR 2019*, 2019.

[38] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[39] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[40] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[41] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[42] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[43] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[44] M. Renardy and R. C. Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[47] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pages 2488–2498, 2018.

[48] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[49] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[50] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[51] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[52] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[54] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

[55] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[56] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. *ICLR, 2019; arXiv preprint arXiv:1812.09926*, 2018.

[57] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[58] F. Xue and J. Xin. Learning sparse neural networks via l0 and tl1 by a relaxed variable splitting method with application to multi-scale curve classification. In *World congress on global optimization*, pages 800–809. Springer, 2019.

[59] B. Yang, J. Lyu, S. Zhang, Y. Qi, and J. Xin. Channel pruning for deep neural networks via a relaxed groupwise splitting method. In *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, pages 97–98. IEEE, 2019.

[60] H. Yang, H. Yin, P. Molchanov, H. Li, and J. Kautz. Nvit: Vision transformer compression and parameter redistribution. *arXiv preprint arXiv:2110.04869*, 2021.

[61] F. Yu, K. Huang, M. Wang, Y. Cheng, W. Chu, and L. Cui. Width & depth pruning for vision transformers. In *AAAI Conference on Artificial Intelligence (AAAI), 2022*, 2022.

[62] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, May 2016.

[63] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[64] S. Zhang and J. Xin. Minimization of transformed L_1 penalty: Closed form representation and iterative thresholding algorithms. *Communications in Mathematical Sciences*, 15(2):511–537, 2017.

[65] S. Zhang and J. Xin. Minimization of transformed L_1 penalty: theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming*, 169(1):307–336, 2018.

[66] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Arik, and T. Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *AAAI Conference on Artificial Intelligence (AAAI), 2022*, 2022.

[67] M. Zhu, Y. Tang, and K. Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

[68] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2020.

[69] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[70] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.