第 8 卷 第 3 期
2019 年 5 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 8 No. 3
May 2019

引文格式：
许强, 徐杨杰, 姜玉林, 等. 一种基于强化学习的限定代价下卷积神经网结构自动化设计方法［J］. 集成技术, 2019, 8(3): 42-54.
Xu Q, Xu YJ, Jiang YL, et al. Automatically designing the cost-constrained convolutional neural network architectures with reinforcement learning［J］. Journal of Integration Technology, 2019, 8(3): 42-54.

一种基于强化学习的限定代价下卷积神经网结构自动化设计方法

许 强[1,2] 徐杨杰[1,2] 姜玉林[1] 张 涌[1,2]

[1](中国科学院深圳先进技术研究院 深圳 518055)
[2](中国科学院大学 北京 100049)

摘 要 目前的神经网络结构自动化设计方法主要对所设计神经网络结构的预测准确率进行优化。然而，实际应用中经常要求所设计的神经网络结构满足特定的代价约束，如内存占用、推断时间和训练时间等。该文提出了一种新的限定代价下的神经网络结构自动化设计方法，选取内存占用、推断时间和训练时间三类代表性代价在 CIFAR10 数据集上进行了实验，并与现有方法进行了对比分析。该方法获得了满足特定代价约束的高准确率的卷积神经网络结构，可优化的代价种类比现有方法更多。

关键词 深度学习；强化学习；卷积神经网；网络结构搜索；代价优化
中图分类号 TP 39 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20190225001

Automatically Design Cost-Constrained Convolutional Neural Network Architectures with Reinforcement Learning

XU Qiang[1,2] XU Yangjie[1,2] JIANG Yulin[1] ZHANG Yong[1,2]

[1](*Shenzhen Institutes of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China*)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049, *China*)

Abstract Recently, automated neural network architecture design (neural architecture search) has yielded many significant achievements. Improving the prediction accuracy of the neural network is the primary goal. However, besides the prediction accuracy, other types of cost including memory consumption, inference time, and training time are also very important when implementing the neural network. In practice, such types of cost are often bounded by thresholds. Current neural architecture search method with budgeted cost constraints can only optimize some specific types of the cost. In this paper, we propose budgeted efficient neural architecture search

收稿日期：2019-02-25 **修回日期**：2019-03-05
基金项目：国家自然科学基金重点项目(61433012)；科技部重点研发计划项目(2018YFB0204005)
作者简介：许强，硕士研究生，研究方向为深度学习及强化学习应用；徐杨杰，硕士研究生，研究方向为深度学习及强化学习应用；姜玉林，硕士研究生，研究方向为深度学习应用；张涌(通迅作者)，博士，研究员，博士研究生导师，研究方向为算法优化、分布式计算、大数据等，E-mail：zhangyong@siat.ac.cn。

(B-ENAS) to optimize more types of cost. The experimental results on the well-adopted CIFAR10 dataset show that B-ENAS can learn convolutional neural network architectures with high accuracy under different cost constraints.

Keywords deep learning; reinforcement learning; convolutional neural network; neural architecture search; cost optimization

1 Introduction

The designing of neural network architecture is a fundamental problem in deep learning. However, this process is mainly done manually. It is a challenging and tedious task to design an efficient network architecture, especially for large scale neural network. Recently, some automated neural network architecture search methods have been proposed, such as reinforcement learning based methods[1-3] and evolution algorithm based methods[4-6], etc. However, these methods only focus on the prediction accuracy of the designed architecture. Usually, the prediction accuracy is not the only aspect that we have to consider. There are some other important types of cost which should be considered in practice, e.g., the memory consumption, the inference time and the training time. For example, for embedded devices with limited memory, the network size that can be tolerated is quite restricted. For systems with high real-time requirements, high computational latency is unacceptable.

For neural architecture search problem with budgeted cost constraints, there is a method called BSN (budgeted super network)[7] that can handle various types of cost, but it is not able to optimize the cost that cannot be known before training. In this paper, we propose a method, say budgeted efficient neural architecture search (B-ENAS). Comparing to BSN, besides the types of cost that can be known before training, e.g., the memory consumption and the inference time, B-ENAS can also optimize the cost that can only be obtained after training, like the training time.

2 Related Work

2.1 Neural architecture search

Recently, researches on automated neural network architecture search (neural architecture search) have come a long way. The current neural architecture search methods can mainly be divided into three types: reinforcement learning based, evolution algorithm based and other methods that are different from the former two. The general idea of reinforcement learning based methods is to firstly define the search space of the network architecture. Then, sample the network architecture in the search space through an agent. Finally, use the accuracy of the sampled architecture as the reward and train the agent by maximizing the cumulative reward. The most representative works are works of Zoph and Le[1] and Baker et al.[8], but these two methods train the sampled architecture from scratch with a very high computation cost, e.g., hundreds to thousands of GPU days. Many interesting approaches have been come up to overcome this drawback. Baker et al.[9] predict the final performance of the partially trained model to stop the training process as early as possible. Brock et al.[10] use a HyperNetwork to generate the weights of a model conditioned on that model's architecture. Jin et al.[11], Cai et al.[12] and Zhong et al.[13] implement network transformations/

morphisms to explore the architecture search space based on the classic networks and reuse the weights instead of starting from scratch. In the work of Zoph et al.[2], the RL agent learns to sample a cell (layer) which is repeatedly concatenated to form a complete model. It is much faster than searching for a complete model and the cell itself is more likely to be generalized to other problems. The idea of learning cells is also given in works of Pham et al.[3], Real et al.[5], Zhong et al.[14] and Liu et al.[15]. ENAS (efficient neural architecture search)[3] is a new method that further improved on NAS (neural architecture search)[1] and NASNet[2]. ENAS is 10X faster and 3X less resource-demanding than the original NAS method through parameter sharing. ENAS also achieves better accuracy than NAS and NASNet.

Another important type of network architecture search method is evolution algorithm based methods[4-6,15]. In addition, Saxena et al.[16] start with training a large network called "convolution neural fabric" and prune it in the end. Negrinho et al.[17] proposed an extensive and modular language for architecture search. It can leverage the structure of search space to introduce different architecture search methods. Liu et al.[18] showed an algorithm called progressive neural architecture search (PNAS) which uses a sequential model-based optimization (SMBO) strategy. Elsken et al.[19] proposed an algorithm called neural architecture search by hillclimbing (NASH) which is based on a simple hill climbing procedure. Recently, being different from conventional approaches that applying reinforcement learning or evolution, Liu et al.[20] formulate the neural architecture search problem in a differentiable manner, which is much faster than those non-differentiable techniques.

2.2 Cost optimization

All of the neural architecture search methods mentioned in the previous part only optimize the accuracy of the designed network. In practice, the prediction accuracy usually is not the only aspect that should be considered. We may care more about other types of cost, e.g., the memory consumption, the inference time and the training time, etc. For instance, in embedded devices with limited memory, we may have a demand like "the model size is less than 50 Mb". For systems with high real-time requirements, we may have a requirement like "the inference time is less than 3 milliseconds". To solve these problems, Dong et al.[21] and Huang et al.[22] optimize the inference time by designing a special network architecture, but these two methods can only be used to optimize the inference time. The most relevant research on neural network cost optimization is the network compression methods, such as Hassibi et al.[23], Vanhoucke et al.[24] and Han et al.[25], but these network compression methods are posterior, that is, it must be done on an obtained model. Can we have a method that can actively choose an architecture that satisfies certain cost constraint in the process of network architecture search? Veniat and Denoyer[7] proposed a method called BSN which tries to find neural network architectures that satisfy certain cost constraints in the searching process. BSN samples the network architecture in a predefined network architecture search space by a parameter distribution, then adds up the cross-entropy loss of the sampled network architecture and the cost to be optimized (e.g., the size of parameters) together as the new loss, finally uses policy gradient method to minimize the new loss to

train the parameters of the parameter distribution and sampled network architecture. Because the parameters of the parameter distribution and sampled network architecture are trained simultaneously, this method cannot optimize the cost (e.g., the training time) which can only be obtained after training the sampled network architecture.

3 Methods

Our method is mainly inspired by the reinforcement learning based neural network architecture search method ENAS[3] which greatly reduces the calculation of the same kind methods NAS[1] and NASNet[2] through parameter sharing. Our method is proposed to solve neural architecture search problem with budgeted cost constraints, so it is named by B-ENAS. These NAS-like methods[1-3] are of the similar framework. They use a recurrent neural network as the controller to sample the network architecture A in a predefined search space. As shown in Fig. 1, the controller uses A's accuracy on the validation set as the reward R and is trained with policy gradient to maximize the expected reward.

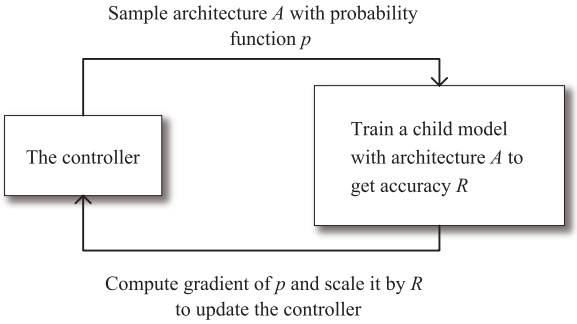

Fig. 1　Procedure of NAS-like algorithms

Note that the final result obtained by the ENAS method is not a single model but a family of models with similar accuracy. These models differ much in cost. The controller is trained by maximizing the expected reward with policy gradient. The reward is the only feedback that the controller needs during the training process. This inspires us whether these different types of cost can be reasonably used to penalize the reward R to optimize the controller and enable it to select network architectures that satisfy certain cost constraints.

Our method B-ENAS is mainly based on ENAS. The main differences of B-ENAS and ENAS are as follows:

(1) Use the cost of the sampled neural network architecture to penalize the reward through a designed *penalty function* to make the controller be able to sample neural network architectures that satisfy certain cost constraints. More details of this point are given in part 3.2.

(2) The number of blocks in a cell is not fixed but decided by the controller, so the search space is bigger and the potential sampled network architectures are more diversified. More details of this point are given in part 3.1.

The main steps of our method are shown in Fig. 2.

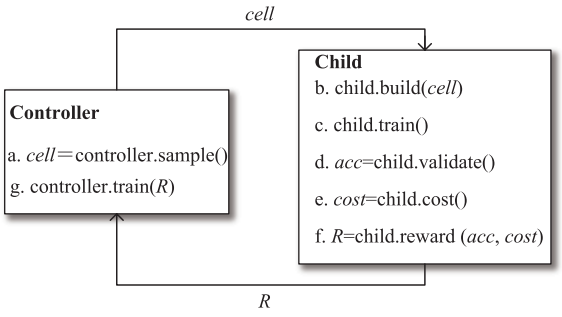

Fig. 2　Procedure of B-ENAS

In step a, the controller samples network architecture in the predefined search space. Note that the controller samples only one cell (layer). We will explain later why we sample a cell rather than a complete model. In step b, assemble a complete

child model using the cell sampled by the controller in step a. In step c, train the child model. In step d, obtain the accuracy *acc* of the child model on the validation set. In step e, calculate the cost of the child model. In step f, use cost in step e to penalize accuracy in step d to get the reward *R*. In step g, use *R* calculated in step f to train the controller.

In our method, the controller and child are trained alternately, in this way, B-ENAS can not only optimize the memory consumption and the inference time, but also the training time, which cannot be known before training the child model. The experimental results on CIFAR10 dataset show the performance of our method in dealing with different types of cost.

In the following parts, we will describe and analyze each component of B-ENAS respectively:

(1) How to sample the network architecture and assemble the sampled architecture into a complete model and train it.

(2) How to use the accuracy and cost to calculate the reward.

(3) How to train the controller.

3.1 Sample cell and train child model

For steps a and b of B-ENAS, we don't directly sample the complete network as many previous methods[1,8] did, but like NASNet[2], BlockQNN[14] and ENAS[3], we sample a small cell and then use this cell to build a complete model. The construction approach is shown in Fig. 3. The advantage of such idea is obvious, it can improve the computational efficiency of the algorithm, and small cells' transferability is better than the complete network[2].

Being different from NASNet and ENAS, the architecture of convolution cell and reduction cell is the same in our method. But the step of

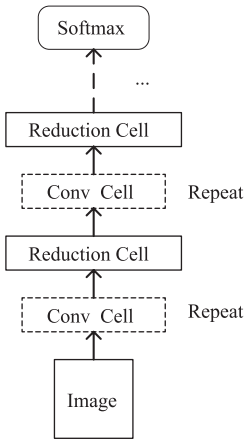

Fig. 3 Build model from cell

convolution and pooling are different. The step size of convolution cell is 1, while the step size of reduction cell is 2. A cell consists of multiple blocks. The number of blocks in each cell of NASNet and ENAS is specified manually. Different from these methods, the number of blocks in B-ENAS is learned by the controller. The basic components of the cell are the same as NASNet and ENAS. Each block in the cell has two inputs, which are labeled as *ind*1 and *ind*2, respectively. Candidate inputs of the current block are the two cells before current cell and blocks before current block in current cell. For example, for a model with 5 cells and each cell has 3 blocks. For the fourth cell's third block, the potential inputs are cell2, cell3, cell4-block1 and cell4-block2. After determining the two inputs of current block, we need to select operations for the two inputs separately. The operations of the two inputs are recorded as *op*1 and *op*2 respectively. There are 5 types of optional operations: identity, separable convolution with kernel size 3×3 and 5×5, average pooling and max pooling with kernel size 3×3. Fig. 4 shows an example of the controller's sampling process and the corresponding cell, where *n* is the maximum number of blocks that can be selected, and *b* represents that

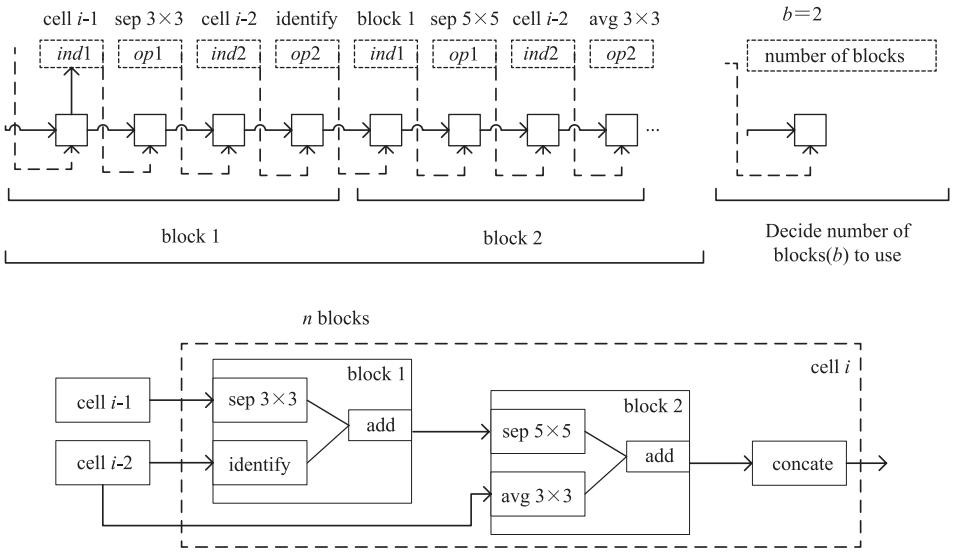

Fig. 4　The controller's sampling process

we select the former b blocks in n to construct the cell. In this example, b=2. The composition of the two blocks and the corresponding cell consisting of these two blocks are shown in Fig. 4, too.

For step c of B-ENAS, we use $p(\partial;\theta)$ to denote the controller's sampling policy, in which θ is the parameters of the controller and ∂ is the search space of cells. As shown in Fig. 2, the learning process of B-ENAS is iterative. Let a epoch denote one update round of the child and the controller. In B-ENAS, epochs executed iteratively until the controller and child converge to a stable state. Note that the search space of cells in different epochs is same, the child models in different epochs may share some parts in common, which is utilized by ENAS to speed up the processing. Let Ω denote the shared parameters of different child models in different epochs. Intuitively, we can assume that current child model initially use parameters of models in former epochs. Thus, the processing can be much faster than learning from scratch. For the shared parameters of the child models, the loss function we need to optimize is the expected loss function $E_{m\sim p(\partial;\theta)}[L(m;\omega)]$, where m denotes the model that containing the cell sampled by the controller in different epochs. Here, Monte Carlo estimation is used as the unbiased estimation of the loss function:

$$E_{m\sim p(\partial;\theta)}[L(m;\omega)]\approx\frac{1}{M}\sum_{i}^{M}L(m_i,\omega_i) \qquad (1)$$

Where m_i is the model containing the cell sampled in epoch i, $\omega_i\in\Omega$ is the parameters of m_i and $L(m_i,\omega_i)$ is the standard cross-entropy loss of the model m_i which is calculated on a mini-batch. According to the work of Williams et al.[26], if we use the SGD (stochastic gradient descent) to update and the learning rate is reasonably selected, the estimation will be eventually converged. Pham et al.[3] showed by experiments that the sampling times M=1 is reasonable.

3.2　Calculate reward

Before the reward calculation (step f), step d and step e should be performed to obtain the prediction accuracy of the current child model and the cost to be optimized on the validation set, respectively.

Actually, these two steps are quite intuitive, so we don't describe them in detail. One of the contribution

of our work is the *reward function*, which is shown as follows:

$$reward = accuracy - \alpha \cdot penalty \tag{2}$$

Where $\alpha \in [0,1]$ is the weighting coefficient, *accuracy* is the prediction accuracy of the child model, *penalty* is the penalty value calculated according to the cost. The *penalty* is calculated through the *penalty function* (Fig. 5):

$$penalty = \begin{cases} 0 & 0 \leqslant c < U_c \\ \beta \dfrac{c}{U_c} + (1-\beta)U_c & U_c \leqslant c < 10U_c \\ 9\beta + 1 & 10U_c \leqslant c \end{cases} \tag{3}$$

Where $\beta > 0$, c is the value of the cost to be optimized, U_c is the upper bound of the cost which is the maximum of the cost we can bear. For instance, in embedded devices with limited memory, we may have a demand like "the model size is less than 50 Mb". In this condition U_c is 50 Mb and notice that the unit measure of c and U_c must match. When c is lower than U_c, there is no *penalty*. When c is bigger than U_c but less than $10U_c$, there is a linear relationship between *penalty* and c, and β is used to adjust the slope and the maximum amount of *penalty*, the relationship between *penalty* and c can also be nonlinear as long as *penalty* increases with c. We cut off the *penalty function* when c exceeds $10U_c$ to prevent extreme values of c from affecting the convergence.

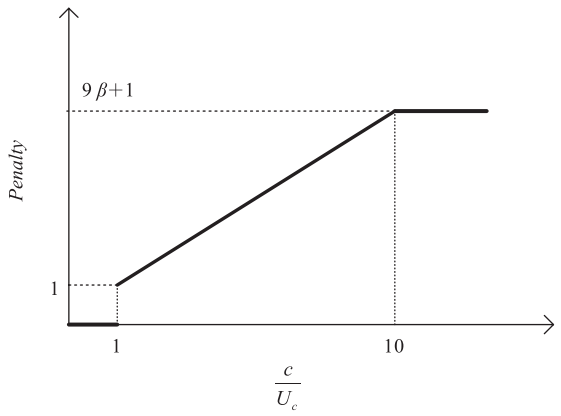

Fig. 5 Penalty function

3.3 Train controller

In B-ENAS, the controller is a 2-layer long short-term memory (LSTM) model. For the last step g, we use policy gradient method to maximize its expected reward to train the controller. The expected reward of the controller is:

$$ER(\theta) = E_{p(\partial;\theta)}[R] \tag{4}$$

Where $p(\partial;\theta)$ is the sample policy of the controller and R is the reward.

The REINFORCE[26-27] algorithm is used to train the controller. The gradient of θ is as follows:

$$\begin{aligned} \nabla_\theta ER(\theta) &= E_{p(\partial;\theta)}[\nabla_\theta \log p(\partial;\theta) R] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p(\partial;\theta)(R_i - avg) \end{aligned} \tag{5}$$

Where n is the number of sampled cells, R_i is the reward of the model containing cell i and *avg* is the reward's exponential moving average which can reduce the variance of the reward.

3.4 Pseudo-code of B-ENAS

In this part, we formally describe each component of B-ENAS. The symbols to be used in the pseudo-code are listed as follows (table 1).

Table 1 Symbols in Pseudo-code

Symbols	Description
D	training dataset
∂	search space of the cell
e	number of epochs
b	number of mini-batches
s	number of steps
η_1	learning rate of the child
η_2	learning rate of the controller
u_c	upper bound of the cost
C	best cell
Ω	sharing parameters of the child
ω	parameters of the child model
θ	parameters of the controller

The main procedure of B-ENAS is shown in Algorithm 1.

Algorithm 1　B-ENAS

Input: D, ∂, e, b, s, η_1, η_2, u_c

Output: C

1: Initialize Ω, θ

2: **for** $i=0$ to $e-1$ **do**

3: $\Omega \leftarrow Trainchild(D, \partial, \Omega, \theta, b, \eta_1)$ /* phase1: algorithm2 */

4: $\theta \leftarrow TrainController(D, \partial, \Omega, \theta, s, \eta_2, u_c)$ /* phase2: algorithm3 */

5: **end for**

6: $C \leftarrow SelectCell(D, \partial, \Omega, \theta)$ /*algorithm4*/

7: return C

During the execution of B-ENAS, an epoch consists of two phases in Algorithm 1. In phase 1, fix the parameters of the controller θ and implement Algorithm 2 (TrainChild) to train the parameters of the child Ω. In phase 2, fix the parameters of the child Ω and implement Algorithm 3 (TrainController) to train the parameters of the controller θ. After e epochs' training, Algorithm 4 (SelectCell) will be executed to select a suitable C that satisfies the constraint. The output C of B-ENAS is the best selected cell, which will be used to build the training model, following the procedure shown in Fig. 3.

In Algorithm 2, we train the child on b mini-batches iteratively. For each mini-batch, we first sample a cell c. Then, use the sampled cell c to build a child model m (as shown in Fig. 3). Finally, train the model m on the current mini-batch by minimizing the cross-entropy loss (as shown in Formula (1) with $M=1$). In this procedure, $\left|D_b\right|$ is the size of current mini-batch.

In Algorithm 3, we train the controller for s rounds. In each round, line 2 to line 9 are executed in order. We first sample a cell c and use this cell c to build a child model m. Then, the model m is validated on a mini-batch to get the accuracy acc. After obtaining model m's cost, the *penalty* will be calculated according to *penalty function* (Formula (3)), the reward R will be calculated according to *reward function* (Formula (2)). We then compute the gradient of the controller's parameters θ by Formula (2). Finally, update the controller's parameters θ.

Algorithm 2　TrainChild

Input: D, ∂, Ω, θ, b, η_1

Output: Ω

1: **for** $i=0$ to $b-1$ **do**

2: $c \sim p(\partial;\theta)$ /*Sample a cell c*/

3: Use *cell* build model m /*Fig. 3*/

4: $\omega \leftarrow \omega - \eta_1 \frac{1}{|D_b|}\sum_{j=1}^{|D_b|}\nabla_\omega L(m;\omega)$

/* $L(m;\omega)$: Formula (1), $M=1$*/

/* ω are parameters of m, $\omega \in \Omega$ */

5: **end for**

6: return Ω

Algorithm 3　TrainController

Input: D, ∂, Ω, θ, s, η_2, u_c

Output: θ

1: **for** $i=0$ to $s-1$ **do**

2: $c \sim p(\partial;\theta)$ /*Sample a cell c*/

3: Use c build model m /*Fig. 3*/

4: $acc \leftarrow$ Validate m on a mini-batch

5: $cost \leftarrow$ Calculate cost of m

6: $penalty \leftarrow$ Input $cost$ and U_c to *penalty function* /*Formula (3)*/

7: $R \leftarrow$ Input acc and $penalty$ to *reward function* /*Formula (2)*/

8: $\nabla_\theta ER(\theta) \leftarrow$ Input $p(\partial;\theta)$ and R to $\nabla_\theta ER(\theta)$ /*Formula (5)*/

9: $\theta \leftarrow \theta + \eta_2 \nabla_\theta ER(\theta)$

10: **end for**

11: return θ

In Algorithm 4, we choose 10 cells and select the most suitable one among them. The selection criteria is the product of the accuracy and the cost instead of only the accuracy. The number of epochs to search is limited due to the search efficiency, but the number of parameters shared between different child models Ω is quite huge. So during the search process, the shared parameters have no need to be fully trained. If

the cost of the sampled cell is larger, that is, the more complex the cell is, the greater impact of the lack of sufficient training is. So the product of the accuracy and the cost can be justified as the selection indicator. Under the premise of satisfying these constraints, the cells with higher value will be selected firstly.

Algorithm 4 SelectCell

Input: D, ∂, Ω, θ

Output: C

1: Initialize $mx \leftarrow 0, C \sim p(\partial; \theta)$
2: **for** $i=0$ to 9 **do**
3: $c \sim p(\partial;\theta)$ /*Sample a cell c*/
4: Use c build model m /*Fig. 3*/
5: $acc \leftarrow$ Validate m on a mini-batch
6: $cost \leftarrow$ Calculate cost of m
7: **if** $acc \times cost > mx$ **then**
8: $C, mx \leftarrow c, acc \times cost$
9: **end if**
10: **end for**
11: return C

4 Experiments and Results

4.1 Experiments

Datasets: Our experiments are carried on the CIFAR-10 dataset. The data preprocessing and augmentation procedures are same as other works in this domain, i.e. subtracting the channel mean and dividing the channel standard deviation, centrally padding the training images to 40×40 and randomly cropping them back to 32×32, and randomly flipping them horizontally.

Search space: The number of blocks of each cell ranges in [2, 7](b in Fig. 4). Each block in the cell has two inputs. The candidate input of the current block contains two cells before the current cell and the blocks before current block in current cell. The five options for these two inputs are identity, separable convolution with kernel size 3×3 and 5×5, average pooling and max-pooling with kernel size 3×3.

Training details: As mentioned before, the B-ENAS can be divided into two phases. In the first phase, only the shared parameters Ω between child models will be updated, while in the second phase, only the parameters θ of the controller will be updated. The setting of the training is similar to ENAS, as shown in table 2.

Table 2 Training details of ENAS

Training details	Child	Controller
Initialization	He initialization	Uniformly in [−0.1, 0.1]
Optimizer	Nesterov Momentum	Adam
Learning rate	Cosine schedule ($l_{max}=0.05$, $l_{min}=0.001$, $T_0=10$, $T_{mul}=2$)	10^{-3}
Others	l_2 weight decay (10^{-4}) Global average pooling	Add temperature(5.0) and tanh(2.5) constant to the logits. Add sample entropy to reward (weighted by 0.1)

Noted that in the searching process of B-ENAS, in order to improve the search efficiency, we use the sampled cell to build a 3-layer model and train it on a mini-batch. At the end of B-ENAS, we use the controller to sample 10 cells, and the cell with the largest product of the accuracy and the cost will be selected to build a 11-layer final model, which will be trained on the whole dataset. The training procedure is the same as in the first phase.

The setting of B-ENAS: Let α in the *reward function* to be 0.1, β in the *penalty function* to be 1. We tested several different values of α and β in the experiments of memory consumption cost and found that B-ENAS is not sensitive to the value of these two super-parameters. 0.1 and 1 is a reasonable setting, so we also took it as the default setting in the experiments of the inference time and the training

time. It is recommended to use this setting as the default setting for B-ENAS. The mini-batch size is 160, b in Algorithm 2 is $2|D|/160$, s in Algorithm 3 is 30, e in Algorithm 1 is 150. In order to improve the training efficiency and transferability, we search for cells instead of the complete model. Therefore, we have to estimate the U_c of the cell according to the U_c of the complete model. Since the final model is stacked by the cell, the U_c of the cell can be estimated based on the U_c of the final model.

4.2 Results

As mentioned before, we consider three important types of cost in the neural network architecture design:

(1) The *inference time*, measured by the number of floating-point operations on predicting the label of a 32×32×3 image.

(2) The *memory consumption*, measured by the size of trainable parameters.

(3) The *training time*, measured by the training time on one Tesla K80 GPU.

The experimental results of BSN[7] are from the original paper.

For the inference time, the comparison of BSN and our method B-ENAS on the inference time is shown in table 3.

BSN can also be used to optimize the memory consumption, the results of BSN and our method B-ENAS on the memory consumption are shown in table 4.

BSN cannot optimize the training time, since the training time can only be obtained after training the sampled child model. Thanks to the interactive learning procedure, our method, B-ENAS, can handle the training time. The results of B-ENAS on the training time is shown in table 5.

Table 3　Results on inference time

Models		Flops (millions)	Accuracy (%)
BSN	B-ResNet	407.51	94.29
		152.60	94.01
		56.47	92.92
		39.25	92.39
	B-CNF	2 150	94.92
		1 407	94.85
		103	93.14
		85	92.17
B-ENAS(ours)		1 040	95.97
		736	96.32
		658	95.85
		539	95.85

Table 4　Results on memory consumption

Models		Parameters (millions)	Accuracy (%)
BSN	B-ResNet	4.38	94.35
		1.29	93.85
		0.34	92.72
		0.29	92.17
	B-CNF	7.56	94.88
		4.98	94.58
		3.67	94.42
		1.19	93.53
B-ENAS(ours)		3.36	96.02
		2.38	95.78
		2.00	95.93
		1.96	95.86

Table 5　Results on training time

Models	Training time (h)	Accuracy (%)
B-ENAS (ours)	92.8	96.06
	65.9	95.86
	57.2	95.33
	36.1	95.49

5 Discussion

Experiments on these three types of cost show that our method is better than BSN on the accuracy. The accuracy of our method does not increase

monotonously with the increased cost as the BSN does. This is because the controller has some randomness when sampling network architectures. And in order to handle the insufficient training problem, we introduce the product of the accuracy and the cost as the selection indicator when determining the final cell. The upper bound of the accuracy is largely determined by the definition of the search space. But in fact, the search space of different methods varies a lot. Even for the same method, the search space will be different on different problems. So the accuracy should not be the only consideration and the ability to satisfy different cost constraints is more important. Through these experiments, we can see that the proposed method B-ENAS can learn the network architecture with different cost constraints as BSN does. Besides the cost that can be known before training the sampled child model, e.g., the inference time and the memory consumption, B-ENAS can also optimize the cost that can only be known after training the sampled child model like the training time.

Consider the cells corresponding to the two models with 1.96 millions and 3.36 millions parameters in the experiments of the memory consumption, as shown in Fig. 6 and Fig. 7, respectively. The cell architecture is simple when the upper bound of the cost is low and is more complicated when the upper bound is higher, which is in line with intuition.

Although B-ENAS has achieved good results in the current experiments, it still has many things to improve:

(1) Because of limited computing resources and time, many important experiments have not been carried out. We have only performed experiments on the CIFAR10 dataset and further experiments can be performed on other datasets such as CIFAR100, SVHN and MNIST. We only experimented with one form of *penalty function*, and we can further experiment with other forms of *penalty function*. The search space does not differ too much from ENAS. If the computing resources are sufficient, the search space can be further expanded to make the sampled network architecture more complicated.

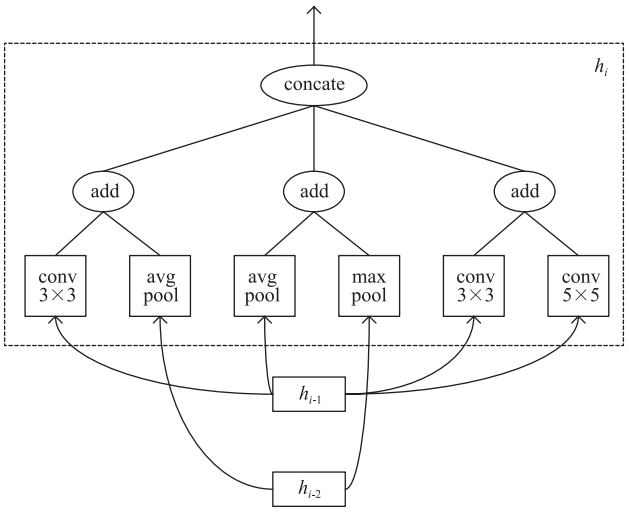

Fig. 6 Cell for model of 1.96 millions parameters

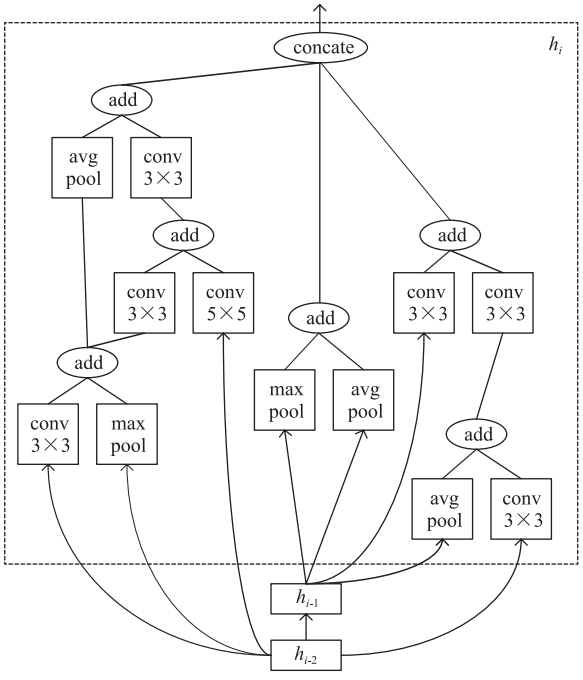

Fig. 7 Cell for model of 3.36 millions parameters

(2) The search space has to be decided manually. The search space has a decisive influence on the result of B-ENAS. If the definition of the search space is unreasonable, so there is no network architecture can satisfy the cost constraint in the search space, then the final result of B-ENAS is certainly not ideal. Therefore, in practical applications, we need to define the search space carefully according to specific problems. This is also a common problem of current neural architecture search methods.

(3) The same as BSN, B-ENAS is also soft constrained, that is, it is not theoretically guaranteed to be able to search for a network architecture that satisfies the cost constraint.

6 Conclusion

In this paper we propose a method B-ENAS to handle the neural architecture search problem with budgeted cost constraints. Especially, to be able to optimize more types of cost than current method BSN. B-ENAS is inspired by current reinforcement learning based neural architecture search method ENAS. The basic idea of B-ENAS is to use different types of cost to penalize the reward so as to make the agent select network architectures that satisfy certain cost constraints. The effectiveness of B-ENAS is verified in experiments on the memory consumption, the inference time and the training time. B-ENAS optimizes these types of cost in the network architecture search process. As mentioned before, the network compression methods do such optimization in a posterior way. Therefore, it is quite interesting to study the combination of the NAS-like methods and the network compression methods in the future.

References

[1] Zoph B, Le QV. Neural architecture search with reinforcement learning [C] // The International Conference on Learning Representations (ICLR), 2017.

[2] Zoph B, Vasudevan V, Shlens J, et al. Learning transferable architectures for scalable image recognition [C] // The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[3] Pham H, Guan MY, Zoph B, et al. Efficient neural architecture search via parameter sharing [C] // The International Conference on Machine Learning (ICML), 2018.

[4] Real E, Moore S, Selle A, et al. Large-scale evolution of image classifiers [C] // The International Conference on Machine Learning (ICML), 2017.

[5] Real E, Aggarwal A, Huang YP, et al. Regularized evolution for image classifier architecture search [C] // The Thirty-Third AAAI Conference on Artificial Intelligence, 2018.

[6] Suganuma M, Shirakawa S, Nagao T. A genetic programming approach to designing convolutional neural network architectures [C] // The Genetic and Evolutionary Computation Conference (GECCO), 2017: 497-504.

[7] Veniat T, Denoyer L. Learning time/memory-efficient deep architectures with budgeted super networks [C] // The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 3492-3500.

[8] Baker B, Gupta O, Naik N, et al. Designing neural network architectures using reinforcement learning [C] // The International Conference on Learning Representations (ICLR), 2017.

[9] Baker B, Gupta O, Raskar R, et al. Accelerating neural architecture search using performance prediction [C] // The Conference on Neural Information Processing Systems (NIPS), 2017.

[10] Brock A, Lim T, Ritchie JM, et al. SMASH: one-shot model architecture search through HyperNetwork [J]. arXiv preprint arXiv: 1708.05344, 2017.

[11] Jin HF, Song QQ, Hu X, et al. Neural architecture search with network morphism [J]. arXiv: 1806.10282v2, 2018.

[12] Cai H, Chen TY, Zhang WN, et al. Efficient architecture search by network transformation [C] // The Association for the Advancement of Artificial Intelligence (AAAI), 2018.

[13] Cai H, Yang JC, Zhang WN, et al. Path-level network transformation for efficient architecture search [C] // The International Conference on Machine Learning (ICML), 2018.

[14] Zhong Z, Yan JJ, Wu W, et al. Practical block-wise neural network architecture generation [C] // The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 2423-2332.

[15] Liu HX, Simonyan K, Vinyals O, et al. Progressive neural architecture search [C] // European Conference on Computer Vision (ECCV), 2018.

[16] Saxena S, Verbeek J. Convolutional neural fabrics [C] // The Conference on Neural Information Processing Systems (NIPS), 2016.

[17] Negrinho R, Gordon G. DeepArchitect: automatically designing and training deep architectures [J]. arXiv: 1704.08792, 2017.

[18] Liu CX, Zoph B, Neumann M, et al. Progressive neural architecture search [C] // The European Conference on Computer Vision (ECCV), 2018: 19-34.

[19] Elsken T, Metzen JH, Hutter F. Simple and efficient architecture search for convolutional neural networks [C] // The International Conference on Learning Representations (ICLR), 2017.

[20] Liu HX, Simonyan K, Yang YM. DARTS: differentiable architecture search [J]. arXiv: 1806.09055, 2018.

[21] Dong XY, Huang JS, Yang Y, et al. More is less: a more complicated network with less inference complexity [C] // The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017: 5840-5848.

[22] Huang G, Chen DL, Li TH, et al. Multi-scale dense convolutional networks for resource efficient image classification [C] // The International Conference on Learning Representations (ICLR), 2018.

[23] Hassibi B, Stork DG. Second order derivatives for network pruning: optimal brain surgeon [C] // The Conference on Neural Information Processing Systems (NIPS), 1993.

[24] Vanhoucke V, Senior A, Mao MZ. Improving the speed of neural networks on CPUs [C] // The Conference on Neural Information Processing Systems (NIPS), 2011.

[25] Han S, Mao HZ, Dally WJ. Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding [C] // The International Conference on Learning Representations (ICLR), 2015.

[26] Williams RJ, Peng J. Function optimization using connectionist reinforcement learning algorithms [J]. Connection Science, 1991, 3(3): 241-268.

[27] Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning [J]. Machine Learning, 1992, 8(3-4): 229-256.