Exploring the Landscape of Differentiable Architecture Search with Reinforcement Learning

Nidhish Shah
Eindhoven University of Technology
n.s.shah@student.tue.nl

Abstract

In recent times, differentiable architecture search (DARTS) has gained significant attention as an effective approach for Neural Architecture Search (NAS). However, DARTS is susceptible to the collapse problem, wherein the search algorithm selects non-parametric operations, resulting in suboptimal architectures. To overcome this limitation, previous studies have employed gradient-based techniques. In this research, we extensively investigate the application of reinforcement learning (RL) to DARTS, exploring various combinations of state space, discrete and continuous action spaces, and three distinct reward functions: `accuracy`, `loss`, and `nwot`. Through experimental evaluations, our RL-based approach achieves a test error of 3.26% on CIFAR-10 within the DARTS search space and demonstrates its robustness in the S2 and S3 RobustDARTS search spaces. The source code is available at: https://github.com/nidhishs/dartsrl.

1 Introduction

Neural Architecture Search refers to the process of automatically discovering optimal neural network architectures for a given task. These architectures have been shown to outperform manually designed neural networks [28, 4, 10]. Prior methods have employed reinforcement learning [28], Bayesian optimization [2], or evolutionary algorithms [17] to tackle this search problem. Traditional approaches that search over a discrete space of candidate operations have proven to be computationally inefficient, often requiring thousands of GPU-hours for the search process. To alleviate this issue, DARTS [11] relaxed the search space to be continuous and dramatically reduced the search time to 2-4 GPU-days.

Differentiable Architecture Search (DARTS) [11] predominantly utilizes gradient based learning approaches, which inherently leads DARTS to display a preference towards non-parametric operations, generally associated with steep gradient decay trajectories [3]. Although an array of solutions have been propose to tackle the collapse [3, 26, 27], these are confined to gradient-based methodologies and offer iterative enhancements to the mechanics of DARTS.

Yet, a notable gap remains unaddressed in the literature —the application of reinforcement learning as a countermeasure to the aforementioned collapse. This study provides an exploratory journey of leveraging reinforcement learning (RL) as a potential solution. Unlike gradient based learning, RL offers a fundamentally different perspective with learning guided by a reward mechanism. With its inherent ability to navigate around premature convergence to a local optima (arising from the exploration-exploitation trade-off), it offers a promising solution for mitigating the undesired bias towards particular operations.

Our objective is to provide a comprehensive and lucid depiction of our approach to applying reinforcement learning to differentiable architecture search. The primary contributions of our research are as follows:

- We propose a framework for conducting differentiable architecture search in both continuous and discrete action spaces. In the continuous space, the agent functions as a meta-optimizer,

using a real-valued vector to update the architecture parameters. Contrarily, in the discrete space, the agent employs sequential assignment of operations to edges within a cell.

- We define the states as the softmax of architecture parameters, optionally including the gradients of these parameters with respect to the validation loss. In addition, we establish three distinct reward functions —`accuracy`, `loss`, `nwot`, each offering a unique measure of performance.
- We corroborate our methodology through rigorous experimentation using different combinations of state space, reward function and environment dynamics. These experiments are performed on the CIFAR-10 [8] dataset within the DARTS search space.
- Finally, we extend our evaluations to examine the robustness of our method by conducting searches in the S2, S3, and S4 search spaces, as proposed in RobustDARTS [27].

2 Preliminaries

In this initial section, our objective is to present a comprehensive overview of differentiable architecture search (DARTS). Additionally, we will introduce the concept of policy optimization, which will establish the necessary foundation for understanding the subsequent integration of DARTS with reinforcement learning.

2.1 Differentiable Architecture Search

Each cell in DARTS is a directed acyclic graph (DAG) of N nodes, where each node $x^{(i)}$ represents a feature map, and each edge (i, j) represents some operation $o^{(i,j)} \in \mathcal{O}$ to transform $x^{(i)}$. Each cell receives the output of the previous two layers as input and concatenates the output of all intermediate nodes to produce the output. Each intermediate node is calculated based on all preceding nodes:

$$x^{(j)} = \sum_{i<j} o^{(i,j)} \left(x^{(i)} \right) \tag{1}$$

The search space is made continuous by relaxing a discrete choice of an operation $o(\cdot) \in \mathcal{O}$ to a softmax over all candidate operations, where $\alpha^{(i,j)}$ is a vector of dimension $|\mathcal{O}|$ representing the operation mixing weights for a pair of nodes (i, j).

$$\bar{o}^{(i,j)} = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x^{(i)}) \tag{2}$$

The discrete operation $o^{(i,j)}$ is obtained by selecting the operation o that maximizes the architecture parameter $\alpha_o^{(i,j)}$, i.e, $o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$.

The objective is to discover the optimal architecture represented by α^* that minimizes the validation loss, denoted as $\mathcal{L}_{val}(w^*, \alpha^*)$. Here, w^* represents the weights associated with the architecture, which are obtained by minimizing the training loss: $w^* = \arg\min_w \mathcal{L}_{train}(w, \alpha^*)$. The aim is to find the architecture that achieves the best performance on the validation set, while the weights are optimized based on the training set. Thus, DARTS can be formulated as a bi-level optimization problem,

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t} \quad & w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \end{aligned} \tag{3}$$

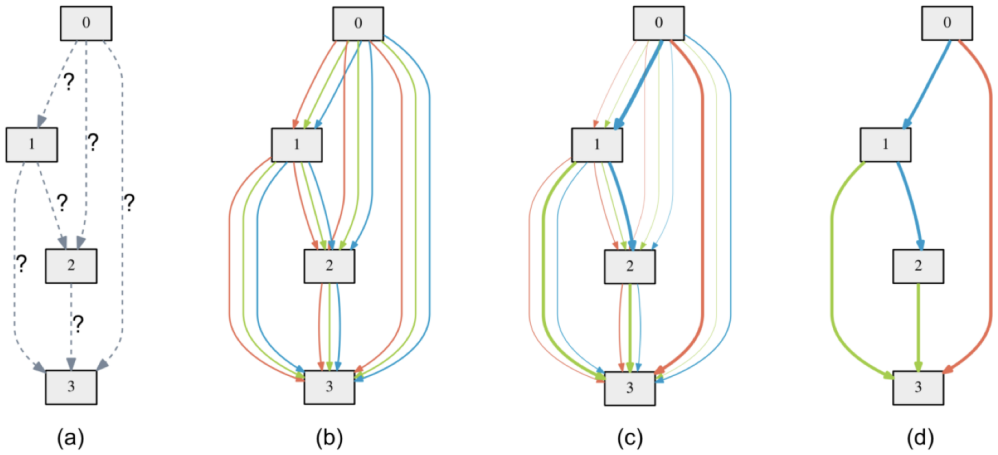


Figure 1: (a) The initial state of edge operations is unknown. (b) The search space is relaxed to allow multiple operations per edge. (c)Optimizing the architecture parameters by solving Eq. 3. (d) Discrete architecture obtained using the probabilities of operations on each edge (image from [11]).

2.2 Policy Optimization

Reinforcement Learning (RL) is a branch of machine learning that focuses on developing computational agents capable of making sequential decisions in dynamic environments. The environment presents a set of states, which the agent observes and selects an action to perform. After each action, the agent receives a scalar reward signal that indicates the desirability of it decision. The goal of RL is to learn an optimal policy (a mapping from states to actions), that maximizes the long term expected cumulative reward [23]. Policy optimization methods have emerged as a prominent and widely adopted approach for training reinforcement learning (RL) agents. Consequently, this section will primarily delve into the subject of policy optimization.

Policy optimization encompasses a family of algorithms that represent a stochastic policy as $\pi_\theta(a \mid s)$, where θ represents the policy parameters, a denotes the action taken within a given state s. These algorithms aim to optimize policy parameters θ by maximizing the expected cumulative reward over trajectories through gradient ascent.

$$\pi_\theta^* = \arg\max_{\pi_\theta} \mathbb{E}_{\tau \sim \pi_\theta} R(\tau) \tag{4}$$

The simplest objective function is typically formulated as,

$$\mathcal{L}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t \mid s_t) A_t \right] \tag{5}$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta \mathcal{L}(\theta) \mid_{\theta_k} \tag{6}$$

where A_t is the advantage function at timestep t. However, the sampling policy and the policy to optimize π_θ are the same in Eq. 5 which can make the learning process unstable. Trust Region Policy Optimization (TRPO) [18] thus used importance sampling and introduced the concept of trust regions by applying a hard KL-divergence constraint between the old policy at previous timestep and the new policy (π_{θ_k} and π_{θ_k} respectively). The constraint stabilizes the learning by preventing excessively large policy updates.

$$\max_\theta \ \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)} A_t \right] \quad \text{s.t} \quad \mathbb{E}_t \left[\mathrm{KL}[\pi_{\theta_k}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right] \leq \delta \tag{7}$$

Motivated by TRPO, Proximal Policy Optimization (PPO) [20] introduced a clipped surrogate objective to constrain the policy divergence, providing a simpler and more practical implementation

while retaining stability during training. The clipped surrogate objective can be formulated as,

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)} A_t \ , \ g(\epsilon, A_t) \right) \right] \tag{8}$$

where,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{for } A \geq 0 \\ (1-\epsilon)A & \text{for } A < 0 \end{cases}$$

3 Methodology

3.1 State Space

Architecture Parameters. The state is composed of the probabilities associated with selecting each operation for every edge in both, the normal and the reduction cells. This probability is determined by applying a softmax function over the architecture parameters corresponding to these cells. Let α_N^i and α_R^i be vectors of dimension $|\mathcal{O}|$, where α_N^i represents the architecture parameters for the i^{th} edge of a normal cell and α_R^i represents the architecture parameters for the i^{th} edge of a reduction cell. The state can then be represented as,

$$s = \left(a_N^1, a_N^2, \ldots, a_N^k, a_R^1, a_R^2, \ldots, a_R^k\right) \tag{9}$$

where k denotes the number of edges in a cell.

Gradients. As an optional feature, the gradients of the architecture parameters with respect to the validation loss are incorporated into the state space. This inclusion presents an opportunity to learn direct relations between parameter adjustments and validation loss. The augmented state space is formulated below:

$$s = \left(a_N^1, \nabla_{a_N^1}\mathcal{L}, a_N^2, \nabla_{a_N^2}\mathcal{L}, \ldots, a_N^k, \nabla_{a_N^k}\mathcal{L}, a_R^1, \nabla_{a_R^1}\mathcal{L}, a_R^2, \nabla_{a_R^2}\mathcal{L}, \ldots, a_R^k, \nabla_{a_R^k}\mathcal{L}\right) \tag{10}$$

Frame stacking. In order to effectively capture temporal trends and variations, we utilize frame-stacking. Specifically, this technique involves concatenating the states from the preceding t time steps to form the state representation at the current time step. This enables the agent to observe patterns in probabilities of candidate operations over time and gain insights into how past architectural decisions influence subsequent states and the overall trajectory of the search process.

3.2 Action Space

Continuous. In the continuous space, the agent acts as a meta-optimizer that optimizes the architecture parameters. Here, the action is a real-valued vector $\nabla\alpha \in \mathbb{R}^{|\mathcal{O}|}$, where $|\mathcal{O}|$ represents the number of candidate operations. In this context, the action serves as updates applied to the architecture parameter equivalent to a gradient descent step, where the direction of the descent is defined by the action vector [9]. The update is formulated below:

$$\alpha^{t+1} \leftarrow \alpha^t + \lambda\nabla\alpha \tag{11}$$

where, λ is a scaling factor, akin to the learning rate of the meta-optimizer.

Each optimization step consists of performing sequential updates, first on the edges of the normal cell, followed by the reduction cell. Each episode consists of performing t such optimization steps. Moreover, to guarantee stable updates, a gradient descent step is performed periodically on a mini-batch of training data with the objective of optimizing the weights of the super network. This step is necessary due to the misalignment between the architecture parameters and the associated super network weights resulting from the optimization of the former.

Although [15] proposes an update similar to Eq. 11, our method exhibits distinct differences. Specifically, our approach incorporates periodic gradient steps, includes all the architecture parameters in the state space, and resets them at the end of each episode.

Discrete. The discrete action space encompasses all the available candidate operations. For each edge in a cell, the agent picks an operation o by taking an action within this discrete space. Thus, the action is represented as a discrete variable i such that $\mathcal{O}[i] = o$.

The environment dynamics in the discrete environment are based on the sequential assignment of operations to edges, first in the normal cell, followed by the reduction cells. Upon taking an action, the weight of the chosen operation is set to a high sentinel value, effectively making this operation the most likely selection for the edge. At the end of each episode, the architecture parameters are reset to their initial values such that each operation has equal probability.

3.3 Reward Function

We employ three distinct reward functions in our environment, namely `accuracy`, `loss`, and `nwot` [14]. The `accuracy` reward function calculates the top-1 accuracy of the network on a mini-batch of validation data. This accuracy is computed on a super network, which may be trained or untrained. Similarly, the `loss` reward function computes the cross-entropy loss of the super network on a mini-batch of validation data, which similar to `accuracy`, can be computed on a trained or untrained super network. Lastly, we utilize `nwot` as a reward function which scores an untrained super network by examining the overlap of activations between data points. Mathematically, it can be formulated as:

$$s = \log |\mathbf{K}_H|, \quad \mathbf{K}_H = \begin{pmatrix} N_A - d_H(\mathbf{c}_1, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(\mathbf{c}_N, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_N, \mathbf{c}_N) \end{pmatrix} \tag{12}$$

where, N_A represents the number of rectified linear units, and $d_H(\mathbf{c}_1, \mathbf{c}_2)$ denotes the Hamming distance between two binary codes. These binary codes are induced by the untrained network at two input points.

The `accuracy` function provides a direct measure of the performance of the architecture and aligns perfectly with the ultimate objective of maximising the accuracy of predictions on unseen data. Similarly, the `loss` function measures how well the network performs in its classification task and provides a precise signal for improvement. Thus, both reward functions are straightforward to understand and compare. On the other hand, we adopt `nwot` for its bias toward convolution operations [7]. Since DARTS favours non-parametric operations, employing `nwot` provides a counteracting force to this collapse.

In terms of reward distribution, it is important to note that intermediate steps yield a zero reward, while the network's score is returned as the reward at the conclusion of each episode. This approach was adopted due to the potential for noisy rewards that might arise from the evaluation of partially optimized networks. By only returning the reward at the end of each episode, we aim to minimize the influence of these noise factors on the learning process.

4 Experiments

We conduct experiments on the CIFAR-10 dataset using the DARTS search space, which consists of eight operations: 3×3 and 5×5 separable convolutions [5], 3×3 and 5×5 dilated separable convolutions [13], 3×3 max pooling, 3×3 average pooling, skip connection and zero operation. The normal and reduction cells have four nodes each, and the edges between nodes can be assigned any of the eight operations. The super network has a fixed depth of 8 cells, with reductions cells at ⅓ and ⅔ of the network. The initial number of channels is set to 16. The super network is trained with half the training data for 50 epochs with a batch size of 64, using the SGD optimizer [22] with

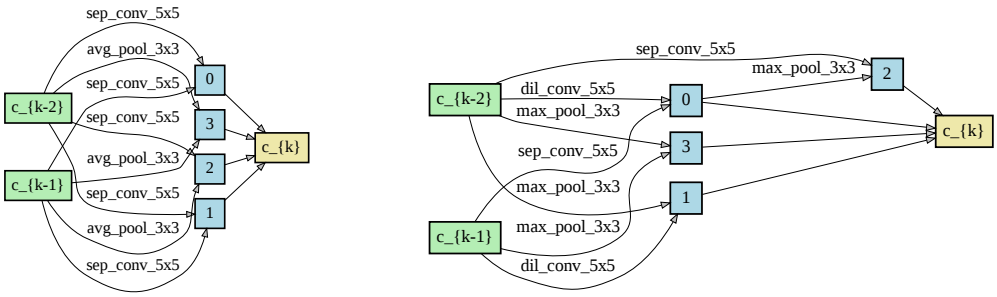
momentum of 0.9 and weight decay of 0.0003. The initial learning rate is set to 0.025 and is annealed down to 0.001 using a cosine schedule [12].

We compare different variants of our method to search the architecture with different settings for the state space, action space, and reward function. We use the Proximal Policy Optimization (PPO) algorithm to train the agent. The network has two hidden layers of size 64 with Tanh activations. The agent is trained with a discount factor of 0.99, a clipping ratio of 0.2, and value function coefficient of 0.5. The agent uses a learning rate of 0.0003 and the Adam optimizer [6] with $\epsilon = 10^{-5}$. We use generalized advantage estimation (GAE) [19] with $\lambda = 0.95$ to compute the advantages.

Finally, we evaluate the target network which has a depth of 20 cells, with 36 initial number of channels. The target network is trained on the entire CIFAR-10 training dataset for 600 epochs with a batch size of 96. Similar to the super network, the target network is optimized using SGD with momentum of 0.9 and weight decay of 0.0003. The learning rate starts from 0.025 and is reduced to zero following a cosine schedule.

We report the results of nine variants of our method, each tested from scratch with three different seeds. The results are summarized in Table 1.

`disc-nwot.` We employ an untrained super network for this experiment. The agent undergoes training for a total of 1,000,000 time steps, with agent optimization taking place every 2048 steps. The action space is discrete, and the state consists solely of the softmax of the architecture parameters. Since the gradients for an untrained network lack meaningful information, we do not incorporate gradients with respect to the validation loss in the state space. Moreover, setting a high sentinel value during each action would cause misalignment of the gradients concerning the architecture parameters. The reward function used is `nwot`. We obtain an average accuracy of 96.74% with a maximum accuracy of 97.01%. The searched architecture can be found in Fig 2.



(a) Normal Cell

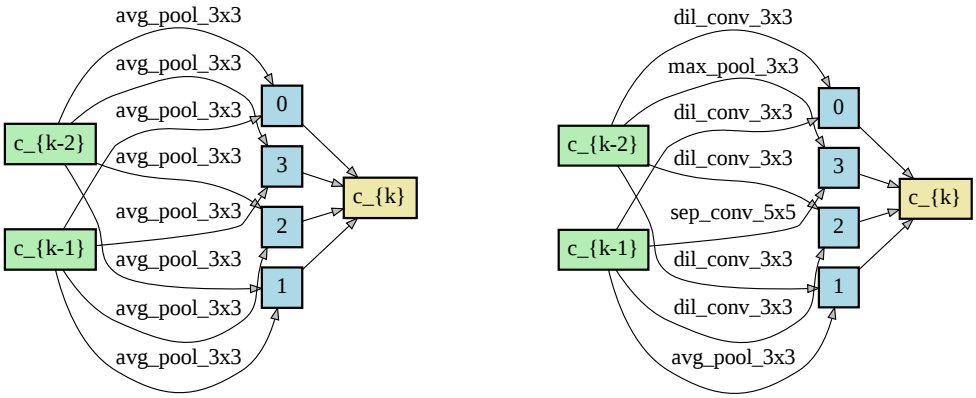
(b) Reduction Cell

Figure 2: Normal and reduction cells searched by `disc-nwot` on CIFAR-10 in DARTS search space.

`disc-acc.` In this experiment, an untrained super network is utilized, and the reward function employed is `accuracy`. All other training conditions remain identical to those of `disc-nwot`. The average performance for this experiment is 93.78%. The searched normal and reduction cells can be found in Fig 3. The inferior performance can be evidently attributed to the normal cell consisting only of pooling layers.

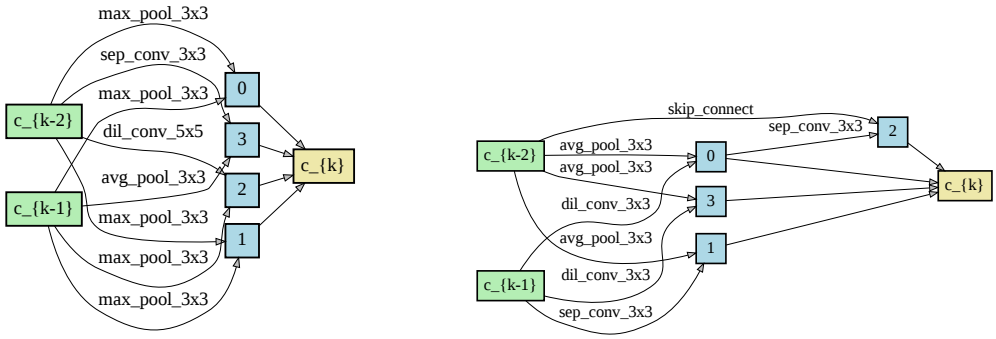
`disc-loss.` This experiment also employs an untrained super network, but the reward function used is `loss`. The remaining training conditions are identical to `disc-nwot`. We achieve an average accuracy of 95.43% and maximum accuracy of 95.49%. Thus we can conclude that `loss` is a better reward function in the discrete action space compared to `accuracy`. The searched architecture is found in Fig 4.

`disc-ckpt-acc.` In contrast to the `disc-acc` experiment, we utilize a super network trained in



(a) Normal Cell (b) Reduction Cell

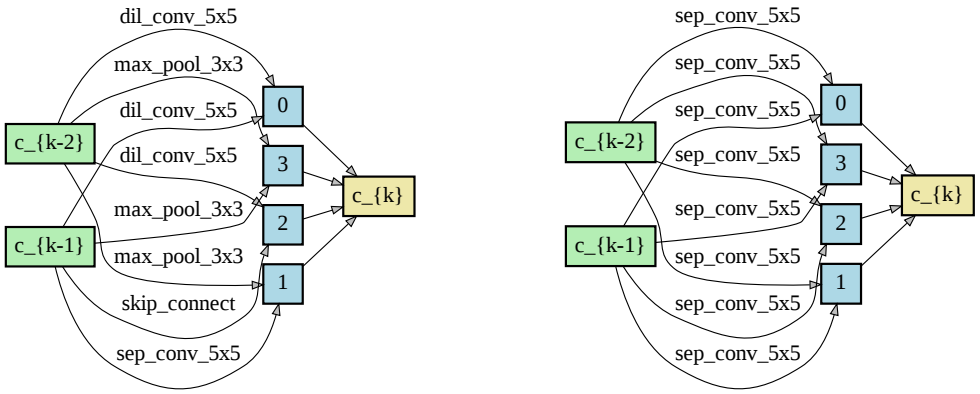
Figure 3: Normal and reduction cells searched by `disc-acc` on CIFAR-10 in DARTS search space.



(a) Normal Cell (b) Reduction Cell

Figure 4: Normal and reduction cells searched by `disc-loss` on CIFAR-10 in DARTS search space.

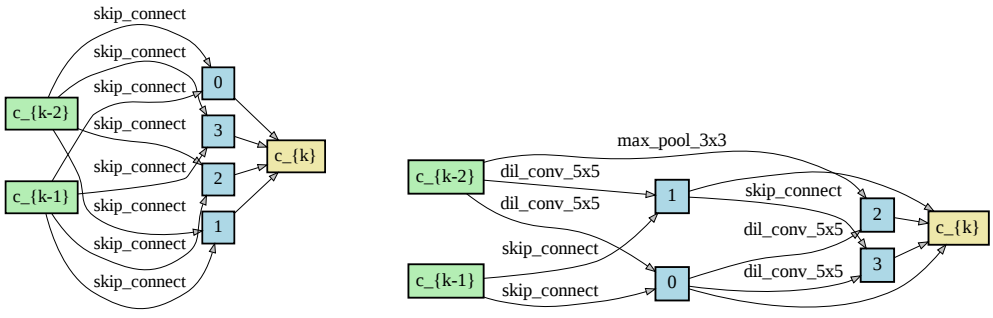
the DARTS search space. All other conditions of the experiment remain unchanged.The average performance achieved is 95.74%. Notably, the utilization of a trained super network for the search process yields improved performance when employing the `accuracy` reward function. The architecture resulting from this search process is illustrated in Figure 5.



(a) Normal Cell (b) Reduction Cell

Figure 5: Normal and reduction cells searched by `disc-ckpt-acc` on CIFAR-10 in DARTS search space.

`disc-ckpt-loss.` We utilize a super network trained in the DARTS search space for this experiment. The other experimental conditions are identical to those of `disc-loss`. The average across three runs is 92.55%. Here, using a trained super network along with the `loss` reward function leads to worse performance compared to `disc-loss`. We theorize that the agent favours non-parametric operations as seen in Fig 6 to minimize the misalignment between super network parameters, architecture parameters and the network predictions.

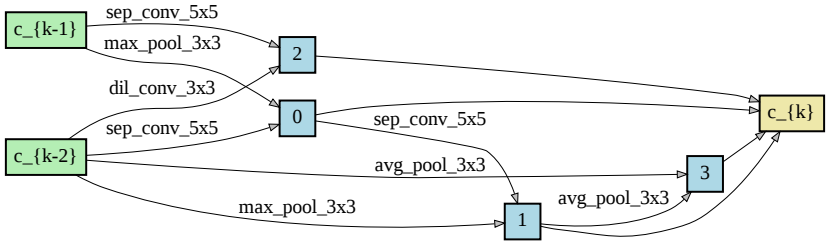


(a) Normal Cell (b) Reduction Cell

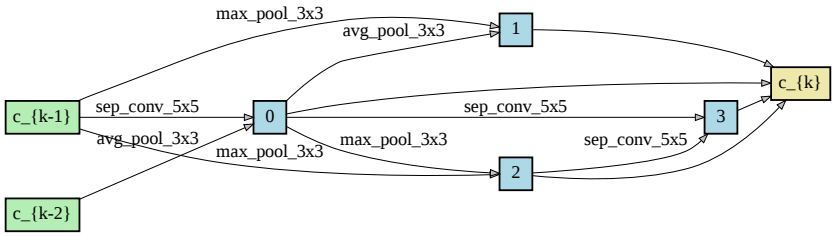
Figure 6: Normal and reduction cells searched by `disc-ckpt-loss` on CIFAR-10 in DARTS search space.

`cont-acc-step.` For this experiment, a trained super network in the DARTS search space is used. The agent is trained using PPO for a total of 250,000 time steps, with agent optimization occurring every 2048 steps. The action space is continuous, and the state is composed of the softmax of the architecture parameters. The updates are scaled by a factor of 0.01. Each episode comprises 10

optimization steps, with the super network being updated through a gradient descent step after every 5 optimization steps. The reward function employed is `accuracy`. We achieve an average performance of 96.08% with 96.35% as the maximum performance. The searched architecture is found in Fig 7.



(a) Normal Cell



(b) Reduction Cell

Figure 7: Normal and reduction cells searched by `cont-acc-step` on CIFAR-10 in DARTS search space.

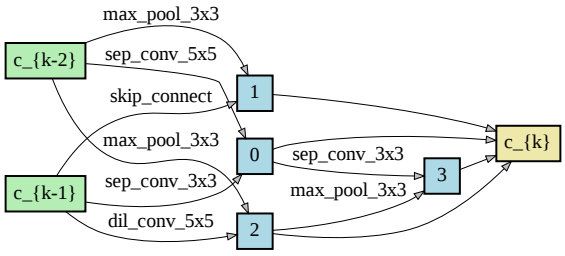
`cont-loss-step.` In line with the `cont-acc-step` experiment, we utilized a trained super network while employing the `loss` function as the reward function. All other training conditions were kept constant. The findings from the `disc-ckpt-acc` experiment can be extended to this context, highlighting that `accuracy` serves as a more effective reward function when utilizing a trained super network. In comparison to the `cont-acc-step` experiment, we observed a reduced average performance of 95.52%. The architecture obtained through the search process is illustrated in Figure 8.

`cont-acc-grad-step.` The training conditions are identical to those of `cont-acc-step`, except that we include the gradients of the architecture parameters with respect to the validation loss in the state alongside the softmax of the architecture parameters. We achieve a slight increase in performance compared to `cont-acc-step` with average performance being 96.24%. This can be attributed to the inclusion of the gradients in the state. The searched architecture can be found in Fig 9.

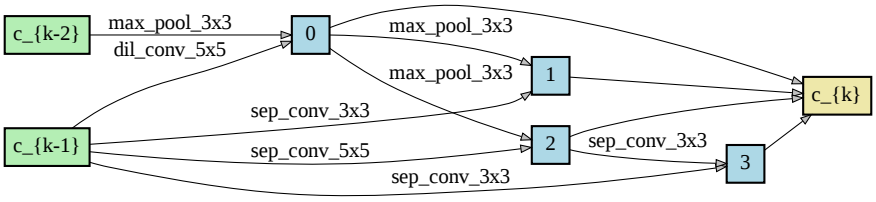
`cont-loss-grad-step.` Similar to `cont-acc-grad-step`, we include the gradients of the architecture parameters in the state and utilize the `loss` reward function. By incorporating the gradients, a slight enhancement in performance is observed, with an average performance of 95.67%.

Our findings indicate that the utilization of a discrete action space combined with the `nwot` reward function leads to optimal performance. To assess the robustness of our algorithm, we conducted searches on three distinct search spaces, namely S2, S3, and S4, as described in [27]. The S2 search space consists of 3×3 separable convolutions and skip connections. The S3 search space builds upon S2 by incorporating a zero operation. Lastly, the S4 search space involves 3×3 separable convolutions and a noise operation.

A robust search algorithm should favor the learnable separable convolution operation in both the S2

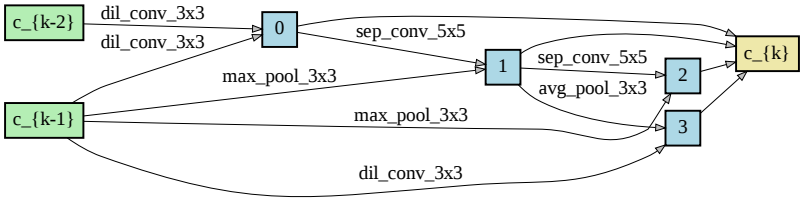


(a) Normal Cell

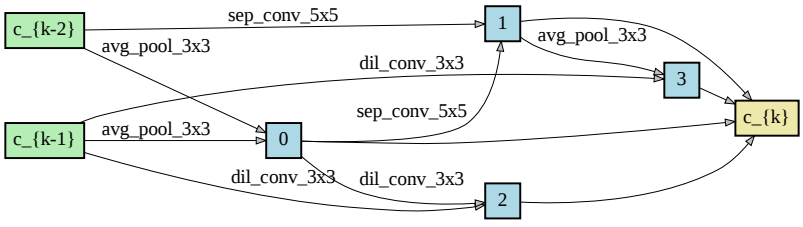


(b) Reduction Cell

Figure 8: Normal and reduction cells searched by `cont-loss-step` on CIFAR-10 in DARTS search space.

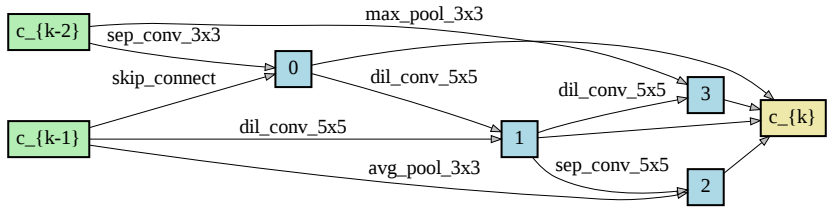


(a) Normal Cell

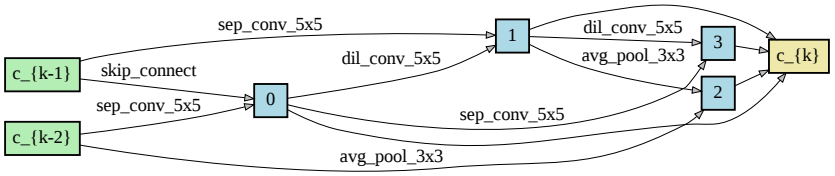


(b) Reduction Cell

Figure 9: Normal and reduction cells searched by `cont-acc-grad-step` on CIFAR-10 in DARTS search space.



(a) Normal Cell



(b) Reduction Cell

Figure 10: Normal and reduction cells searched by `cont-loss-grad-step` on CIFAR-10 in DARTS search space.

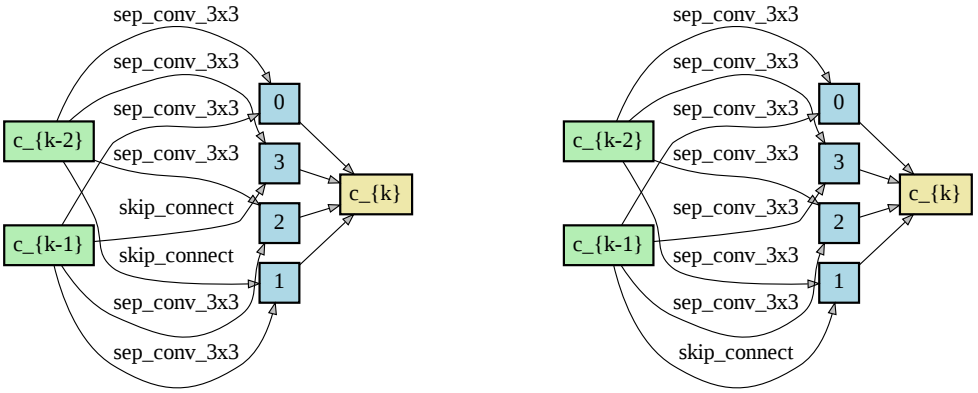
Name	Mean	Max
`disc-nwot`	96.74 ± 0.30	97.01%
`disc-acc`	93.78 ± 0.12	93.94%
`disc-loss`	95.43 ± 0.06	95.49%
`disc-ckpt-acc`	95.74 ± 0.14	95.87%
`disc-ckpt-loss`	92.55 ± 0.39	92.98%
`cont-acc-step`	96.08 ± 0.19	96.35%
`cont-acc-grad-step`	96.24 ± 0.32	96.42%
`cont-loss-step`	95.52 ± 0.16	95.70%
`cont-loss-grad-step`	95.67 ± 0.29	96.00%

Table 1: Search results on CIFAR-10 in the DARTS search space. We report the average results across three runs.

and S3 search spaces, while actively avoiding the noise operation in S4, which has a detrimental effect on performance. Similar to the DARTS search space, we individually train the super network on all three search spaces for 50 epochs with a batch size of 64. We employ the SGD optimizer [22] with a momentum of 0.9 and weight decay of 0.0003. The initial learning rate is set to 0.025 and is reduced to 0.001 through the use of a cosine schedule [12]. The cells discovered by our algorithm are depicted in Figure 11, 12, and 6. It can be observed that by employing the `nwot` reward function, we effectively counter DARTS' inclination towards non-parametric operations in the S2 and S3 search spaces. However, the algorithm also selects the noise operation in the S4 search space, thereby highlighting its lack of robustness across all search spaces.

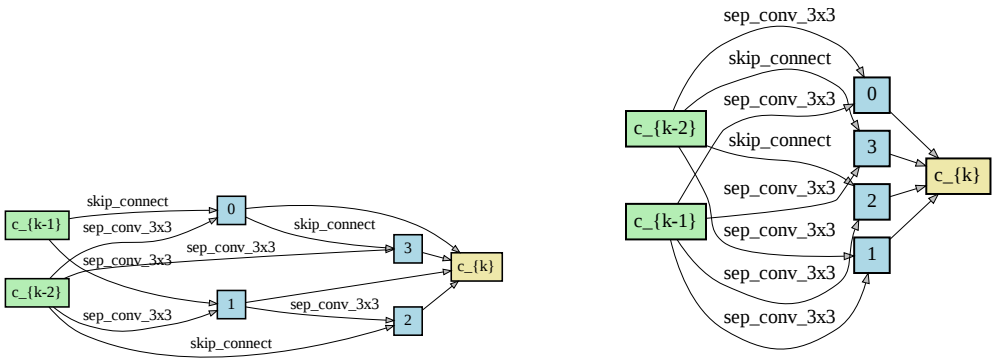
5 Conclusion

In this paper, we have presented a methodology for applying reinforcement learning to differentiable architecture search (DARTS). Our methodology primarily revolves around defining the state space, action space, and reward function within the architecture search environment. We carefully consider



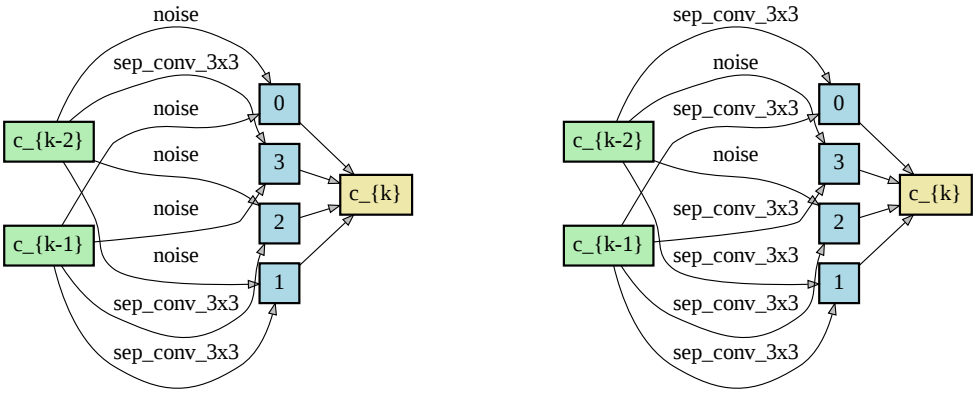
(a) Normal Cell (b) Reduction Cell

Figure 11: Robustness experiment on CIFAR-10 using `disc-nwot` in the S2 search space.



(a) Normal Cell (b) Reduction Cell

Figure 12: Robustness experiment on CIFAR-10 using `disc-nwot` in the S3 search space.



(a) Normal Cell (b) Reduction Cell

Figure 13: Robustness experiment on CIFAR-10 using `disc-nwot` in the S4 search space.

Methods	Test Err.(%)	Search Cost (GPU-days)	Search Algorithm
NASNet-A [29]	2.65	1800	RL
ENAS [16]	2.89	0.5	RL
DARTS (1st order) [11]	3.00 ± 0.14	1.5	Gradient
DARTS (2nd order) [11]	2.76 ± 0.09	4	Gradient
RobustDARTS [27]	2.95 ± 0.21	1.6	Gradient
PC-DARTS [25]	2.57 ± 0.07	3.6	Gradient
DARTS+PT [24]	2.61 ± 0.08	3.0	Gradient
AGNAS [21]	2.53 ± 0.003	3.6	Gradient
DISC-NWOT	$\mathbf{3.26 \pm 0.30}$	**0.08**	**RL**

Table 2: Search results on CIFAR-10 and comparison with other search methods.

both continuous and discrete actions, and we employ diverse reward functions to effectively guide the search process.

Limitations. Although our method shows promising results in terms of robustness, it does not achieve state-of-the-art performance. There is still room for improvement in terms of the search algorithm and the optimization process. Second, the effectiveness of our approach is highly dependent on the chosen reward function. To explore alternative options, we suggest testing other reward functions, such as the zero-cost proxies proposed in [1].

Additionally, our methodology may not generalize well in terms of robustness to all search spaces. The performance and effectiveness of the reinforcement learning algorithm can vary depending on the specific architecture search problem. Therefore, further investigation and experimentation are needed to evaluate the generalizability of our approach across different search spaces.

Broader Impact. By evaluating various combinations of state space, action space, and reward functions, we hope to contribute to the development of more robust and efficient architecture search methods. We believe that our extensive set of experiments serves as a foundation for further exploration and experimentation in the application of reinforcement learning to differentiable architecture search.

References

[1] M. S. Abdelfattah, A. Mehrotra, L. Dudziak, and N. D. Lane. Zero-Cost Proxies for Lightweight NAS. In *International Conference on Learning Representations (ICLR)*, 2021.

[2] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[3] 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.

[4] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun. Detnas: Backbone search for object detection. In *Advances in Neural Information Processing Systems*, 2019.

[5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1800–1807. IEEE Computer Society, 2017.

[6] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[7] A. Krishnakumar, C. White, A. Zela, R. Tu, M. Safari, and F. Hutter. Nas-bench-suite-zero: Accelerating research on zero cost proxies. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 28037–28051. Curran Associates, Inc., 2022.

[8] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research), 2009.

[9] K. Li and J. Malik. Learning to optimize. *CoRR*, abs/1606.01885, 2016.

[10] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019.

[11] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[12] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[13] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[14] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, 2021.

[15] D. Pang, X. Le, and X. Guan. Rl-darts: Differentiable neural architecture search via reinforcement-learning-based meta-optimizer. *Knowledge-Based Systems*, 234:107585, 2021.

[16] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, 10–15 Jul 2018.

[17] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[19] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] Z. Sun, Y. Hu, S. Lu, L. Yang, J. Mei, Y. Han, and X. Li. Agnas: Attention-guided micro and macro-architecture search. In *International Conference on Machine Learning*, pages 20777–20789. PMLR, 2022.

[22] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[23] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[24] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representations (ICLR)*, 2021.

[25] 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.

[26] A. Yang, P. M. Esperança, and F. M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.

[27] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

[28] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[29] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8697–8710. Computer Vision Foundation / IEEE Computer Society, 2018.