DA060492 Course Project: Differentiable Architecture Search

Darya Voronkova [1]

Abstract

Neural Architecture Search is a part of the AutoML field where a large number of various algorithms have emerged in recent years. One of the remarkable approaches is DARTS (Liu et al., 2019b), differentiable architecture search. In this work (Liu et al., 2019b), the authors managed to relax the discrete search space to the continuous one and applied gradient-based optimization to architecture cell search. The models learned with DARTS achieve competitive performance on several tasks while requiring sufficiently reduced computational cost. There are a lot of works improving some weaknesses of DARTS towards more accurate and efficient NAS algorithms. The goal of this project is to implement DARTS algorithm.

1. Introduction

The great success of deep learning models in various fields (Hsu et al., 2021), (Dosovitskiy et al., 2021) is often considered to be the result of automated process of feature engineering that replaced manual feature selection. At the same time, the research during the recent years showed the key importance of the deep architecture itself, that usually involves a huge amount of human efforts and expertise (Elsken et al., 2019), (Ren et al., 2020). Hence, the next level of abstraction is to automatize the process of neural architecture construction. The goal of Neural Architecture Search (NAS) is to automatically construct a neural architecture with high quality under the limited resources. The general pipeline of NAS is shown in the Figure 1. In recent years, a large number of NAS algorithms have emerged and achieved performance competitive with hand-crafted architectures. (Pham et al., 2018), (Wu et al., 2019), (Wen et al., 2020). However, one of the main problems of various approaches in this field is the cost of learning an architecture, i.e. computational time and resources (Liu et al., 2019b). Thus, a lot of works is aimed to make the process of architecture search more efficient. These approaches include well-designed search space (Liu et al., 2018b), weight inheritance (Cai et al., 2018), weight sharing (Pham et al., 2018). A fundamentally different approach is proposed in the paper (Liu et al., 2019b). The general idea of DARTS (Liu et al., 2019b) is to make continuous relaxation of the search space, which allows to formulate the problem of NAS as differentiable task that can be solved via gradient-based optimization. The elegant solution proposed in DARTS algorithm is further developed in a series of subsequent works (Jiang et al., 2019), (Chen et al., 2019), (Dong & Yang., 2019), that obtain better results in terms of both task quality and computational cost.

[1]Skoltech. Correspondence to: Darya Voronkova <Darya.Voronkova@skoltech.ru>.

1.1. General Problem Statement

The goal of this project is to implement the DARTS method and evaluate the learned model. More precisely, the project objectives include the choice of an appropriate convolutional architecture and a reasonable search space, implementations of one-level and bi-level optimization and evaluation of the learned architecture on a dataset of small size (e.g. CIFAR10, MNIST etc.).

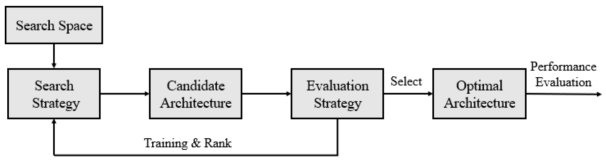

Figure 1. The general framework of NAS. NAS generally starts with a set of predefined operation sets, and uses search strategies to obtain a large number of candidate network architectures based on the search space formed by the operation sets. The candidate network architecture is trained and ranked. Then, the search strategy is adjusted according to the ranking information of the candidate network architecture, thereby further obtaining a set of new candidate network architectures. When the search is terminated, the most promising network architecture is used as the final optimal network architecture, which is used for the final performance evaluation (Ren et al., 2020).

2. DARTS: Differentiable Architecture Search

In this section, we describe the main aspects of the DARTS algorithm (Liu et al., 2019b): search space, continuous relaxation and optimization, approximate architecture gradient. This section is based on the formulation of DARTS algorithm according to the original paper (Liu et al., 2019b).

2.1. Search Space

Following the previous works (Zoph et al., 2018), the authors of DARTS search for a computation cell as a structural element that forms the final architecture. It is possible to learn the cell to obtain a convolutional or a recurrent neural network. A cell is a directed acyclic graph (DAG) with N nodes: each node $x^{(i)}$ represents a latent representation, an operation $o^{(i,j)}$ that transforms $x^{(i)}$ is represented as an edge (i, j). It is assumed that the cell has two input nodes, and the output of the cell is a concatenation of all the intermediate nodes. An intermediate node can be computed using the outputs of previous nodes:

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

2.2. Continuous Relaxation and Optimization

The authors choose a set O of candidate operations (e.g., convolution, max pooling, etc.). To have continuous search space, instead of choosing one operation at a time, the authors propose to have a softmax over all possible operations (so called "mixed" operation):

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

The goal of DARTS is to learn the parameters $\alpha = \left\{\alpha^{(i,j)}\right\}$, where $\alpha^{(i,j)}$ has dimension $|O|$ for each pair (i, j). When the training process of the cell is finished, final discrete architecture of the cell is obtained by replacing each mixed operation with the most probable operation according to the learned parameters α, i.e. $o^{(i,j)} = \arg\max_{o \in O} \alpha_o^{(i,j)}$.

DARTS jointly learns the architecture α and the weights w of operations. DARTS algorithm solves the bi-level optimization problem:

$$\min_{\alpha} L_{val}\left(w^\star(\alpha), \alpha\right) \quad s.t. \quad w^\star(\alpha) = \arg\min_{w} L_{train}(w, \alpha)$$

2.3. Approximate Architecture Gradient

Computation of architecture gradient is prohibitively expensive. Thus, the authors propose the following approximation

$$\nabla_\alpha L_{val}(w^\star(\alpha), \alpha) \approx \nabla_\alpha L_{val}(w - \xi \nabla_w L_{train}(w, \alpha), \alpha)$$

where w denotes the weights on the current iteration, and ξ is the learning rate for a step of inner optimization. According to the chain rule, the approximate architecture gradient can be written as

$$\nabla_\alpha L_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} L_{train}(w, \alpha) \nabla_{w'} L_{val}(w', \alpha)$$

where $w' = w - \xi \nabla_w L_{train}(w, \alpha)$. Sine the expression above involves an expensive matrix-vector product, the authors propose to approximate it with finite difference:

$$\nabla^2_{\alpha,w} L_{train}(w, \alpha) \nabla_{w'} L_{val}(w', \alpha) \approx$$

$$\frac{\nabla_\alpha L_{train}(w^+, \alpha) - \nabla_\alpha L_{train}(w^-, \alpha)}{2\varepsilon}$$

where ε is a small scalar and $w^\pm = w \pm \varepsilon \nabla_{w'} L_{val}(w', \alpha)$. This approximation reduces the complexity from $O(|\alpha||w|)$ to $O(|\alpha| + |w|)$.

When $\xi = 0$, the architecture gradient is given by $\nabla_\alpha L_{val}(w, \alpha)$. The authors note that while it leads to some acceleration in practice, its performance is worse. This setup is referred to as first-order approximation.

3. Literature Review

One of the most popular categorizations of the NAS research is given by the work (Elsken et al., 2019). According to (Elsken et al., 2019), mostly the NAS models vary in the three components: *search space*, *search strategy*, *performance estimation strategy*.

Search space determines the set of architectures that can be represented and learned by the particular NAS method. According to (Elsken et al., 2019), there are several typical choices of search spaces: chain-structured neural networks, multi-branch networks (Cai et al., 2018), networks consisting of repeated cells (Liu et al., 2019b).

Search strategy defines how to investigate the search space solving the conventional problem of "exploration vs exploitation". The main popular directions for search strategy are reinforcement learning (RL) (Zoph & Le, 2017), evolutionary algorithms (Real et al., 2018) and gradient-based optimization (Liu et al., 2019b), (Wu et al., 2019).

Performance estimation strategy refers to the task of candidate model's performance estimation on unseen data. The straightforward yet computationally expensive approach is to train the model from scratch and evaluate it on validation set. In practice, there are methods of reduced computation cost such as partial training (training for fewer epochs, on subset of data, etc.) (Zoph et al., 2018), weight sharing (candidate models are subgraphs of one large network) (Pham et al., 2018).

(Zoph & Le, 2017), one of the pioneering works in NAS, is based on reinforcement learning. In this approach, an agent

builds the architecture sequentially layer by layer while the reward is the validation accuracy after training the model from scratch. The authors show that the learned models achieve performance comparable with the best performing networks, but the computational cost of search is extremely high.

In evolutionary search (Real et al., 2018), there is a set of population models and in each step, a parent model is sampled from population, and undergoes mutations to produce child networks. Mutations are some local operations (e.g., adding a layer of skip-connection) and the child networks are added to the population based on their estimated performance. The existing methods based on evolutionary search vary on the procedures of sampling parents, generating child networks and updating population (Elsken et al., 2019). The work (Real et al., 2019) compared the performance of RL-based algorithms and evolutionary algorithms, revealing that they perform almost equally in terms of test accuracy.

One of the remarkable works in gradient-based optimization is DARTS (Liu et al., 2019b). But there are other approaches towards continuous search space. For example, (Xie et al., 2019), (Cai et al., 2019) use a parameterized distribution over the set of operations to perform gradient-based optimization, while (Shin et al., 2019) makes the relaxation similar to (Liu et al., 2019b) but over the set of a layer's hyperparameters.

An important step in development of NAS is weight sharing trick (Bender et al., 2018) where the candidate models are sub-graphs of a larger model. The weights are shared and trained simultaneously. This technique is often used in different approaches, for example, as (Pham et al., 2018) and (Liu et al., 2019b).

A separate direction in NAS is to predict the model's performance without training. For example, in TE-NAS (Chen et al., 2021) the authors exploit the eigenvalues in neural tangent kernel and the number of linear regions in the input space to construct the best performing model.

Recent research finds application of various algorithms to the NAS. For example, in AlphaNet (Wang et al., 2021), the authors incorporate knowledge distillation with α-divergence between the super-net as a teacher and sub-network as student into the training process. (Wang et al., 2021) with modifications show impressive performance according to (https://paperswithcode.com/sota/neural-architecture-search-on imagenet).

3.1. DARTS: Pros and Cons

By the time the DARTS algorithm was proposed, the NAS had already reached the state-of-the-art performance in image classification task (Zoph et al., 2018), (Real et al., 2018). However, the computational cost to obtain those solutions was extremely high. For example, as argued by the authors of (Liu et al., 2019b), NAS with reinforcement learning (Zoph et al., 2018) required 2000 GPU days while the evolutionary algorithm (Real et al., 2018) required 3150 GPU days. High computational cost presents a severe problem for research in NAS field, especially in case of limited resources or newcomers with little expertise. There are several works that propose to accelerate computations with various techniques such as well-designed search space (Liu et al., 2018b), inheritance of weights during training (Cai et al., 2018) or weight sharing (Pham et al., 2018). However, the main reason for scalability problem in many NAS approaches is that they mostly work in the statement of black-box optimization over a discrete domain. Thus, while the proposed approaches (Liu et al., 2018b), (Cai et al., 2018), (Pham et al., 2018) reduce computational burden having competitive or outperforming quality, they do not solve the problem in general. On the contrary, DARTS proposes a way to relax the discrete search space to be continuous which allows to replace the complicated black-box optimization with efficient gradient-based optimization. As a result, one of the main advantages of DARTS is sufficient reduction of computational cost while preserving highly competitive performance: it achieved high quality on CIFAR-10 dataset, competitive with the state-of-the-art performance of regularized evolution algorithm (Real et al., 2018) using three orders of magnitude less resources. Moreover, the authors (Liu et al., 2019b) showed high quality of DARTS approach when transferred a convolutional cell learned on CIFAR-10 dataset to ImageNet.

The work (Shin et al., 2019) explores the similar idea to relax the discrete search space to continuous form allowing differentiation. However, they do this not for the cell architecture itself but for hyperparameters of the layers, with main focus on hyperperameters of convolutional layers (i.e. filter size, number of channels, etc.). This may be seen as another advantage of DARTS that optimizes the structure of architecture cell and hence works with more complex and powerful search space.

There was another set of models (Saxenaa & Verbeek., 2016), (Ahmed & Torresani., 2017), (Veniat & Denoyer., 2018) that also followed the idea of search over continuous set but they restricted the architecture choice to some specific class. Thus, another important advantage is that DARTS is not limited to any particular model choice, and it is possible to learn the cell for both convolutional and recurrent network as a final architecture.

The work (Ren et al., 2020) defines some of the main disadvantages of the DARTS method. There are several approaches that attempt to eliminate the observed deficiencies of DARTS.

I-DARTS (Jiang et al., 2019). In DARTS approach, a node

may be connected to several previous nodes. However, it is not possible to compare edges coming from different nodes and there should be only one edge between each pair of nodes in the final discretized architecture cell. This local mode of DARTS may result in sub-optimal solution. In the (Jiang et al., 2019) paper the authors propose to use softmax over all incoming edges for a particular node, making it possible to compare all the incoming edges simultaneously. In practice, this allows to outperform DARTS with better convergence and faster training.

P-DARTS (Chen et al., 2019). Another known issue of DARTS is the "depth gap" (Chen et al., 2019) between the architectures during training and evaluation. Due to limited GPU memory, DARTS searches the cell structure in a shallow network (e.g., 8 cells) and evaluates the performance in a deeper networks (e.g., 20 cells). Since the cell was optimized in a smaller architecture, it may be no longer optimal for a deeper network. To alleviate this problem, the authors (Chen et al., 2019) propose P-DARTS to progressively increase the depth of the network during training of the cell. Having lower depth at the beginning of training allows to eliminate the operations with small weight on the later iterations. This helps to overcome the problem of increase in search space size with depth growth. According to the experiments, this approach achieves lower classification error on CIFAR10 and CIFAR100 and requires less amount of computational resources.

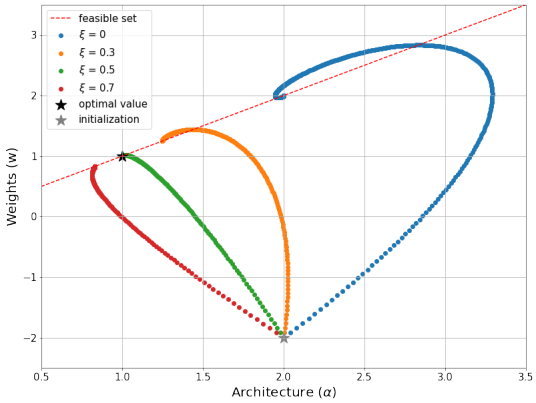

Figure 2. Reproducibility result for the toy experiment on bilevel optimization from the original paper (Liu et al., 2019b) (see Figure 2 in the original paper).

GDAS (Dong & Yang., 2019). In the (Dong & Yang., 2019), the authors point out two weaknesses of DARTS. First, DARTS uses outputs of all operations at each iteration and the cost to update all the parameters simultaneously is high. This also influences the total training time. Second, in simultaneous optimization, different operations can compete with each other, and this may lead to unstable optimization procedure. The GDAS (Dong & Yang., 2019) approach proposes to sample one sub-graph of the DAG at each iteration. In this case, each optimization step influences only part of a DAG, reducing unnecessary competition and accelerating the training process. According to the experimental results (Dong & Yang., 2019), GDAS is able to learn the model with performance comparable to DARTS but reduced computation cost.

PC-DARTS (Xu et al., 2020). High memory consumption is another known issue of the DARTS algorithm. To alleviate this problem, PC-DARTS (Xu et al., 2020) approach samples the channels and convolves the sampled channel features. Instability may be a side effect of this procedure, thus, (Xu et al., 2020) also applies edge normalization to make the optimization more efficient. PC-DARTS sufficiently outperforms the baseline DARTS both in terms of search time and accuracy.

DARTS aroused interest in the NAS community, and there are more methods based on DARTS (Li et al., 2020), (Chen & Hsieh., 2020) or similar ideas (Wu et al., 2019), (Cai et al., 2019), (Liu et al., 2019a).

4. Experiments

4.1. Example of bilevel optimization

First, we reproduce the toy experiemnt on bilevel optimization. Following the experiment from the original paper (Liu et al., 2019b) (see Figure 2 in the original paper), we use:

$$L_{\text{train}}(w, \alpha) = w^2 - 2\alpha w + \alpha^2, \tag{1}$$

$$L_{\text{val}}(w, \alpha) = \alpha w - 2\alpha + 1, \tag{2}$$

where w is the lower-level variable and α is the upper-level variable. Optimization starts from the initialization $(\alpha^{(0)}, w^{(0)}) = (2, -2)$. Analytical solution to this bilevel optimization problem is $(\alpha^*, w^*) = (1, 1)$. Also, there is a feasible set (given by equation $w(\alpha) = \alpha$) where the constraint on w holds exactly. We follow the algorithm of approximate bilevel optimization presented in the original paper (Liu et al., 2019b) (for simplicity, we used gradient descent without momentum) and reproduce the results. In this experiment, we use the learning rate $1e-2$, maximal number of iterations 10000, the threshold $t = 1e-4$. Optimization stops when either the maximal number of iterations is exceeded or both the deviations of w and α on the adjacent iterations is smaller than threshold: $|\alpha^{(i+1)} - \alpha^{(i)}| < t$ and $|w^{(i+1)} - w^{(i)}| < t$.

Similar to (Liu et al., 2019b), we plot the optimization trajectories for several choices of the hyperparameter ξ, see Figure 2. Also, we estimate the optimization time / number of iterations to reach stop criteria for each value of ξ, see Table 1.

Table 1. Estimate of computational cost in terms of number of iterations (out of 10000) and time per iteration (sec).

	$\xi = 0$	$\xi = 0.3$	$\xi = 0.5$	$\xi = 0.7$
Number of iterations	659	411	296	335
Time per iteration	$4.4e-4$	$9.0e-4$	$9.1e-4$	$9.3e-4$

We can note that for all values of ξ the algorithm converges at least to the feasible set. Optimal value of ξ seems to be close to 0.5 and results in the shortest optimization trajectory (in terms of number of iterations) and most accurate solution. Moving away from the optimal value of ξ, we obtain longer convergence and less accurate results. From Table 1 we obtain that time per iteration for the first-order approximation (i.e. $\xi = 0$) is almost twice smaller than for the second-order approximation ($\xi > 0$). Our results are consistent with the paper (Liu et al., 2019b) outcomes.

4.2. Neural Architecture Search

In this section we describe how the neural architecture search is performed. First, we discuss the set of operations and architectures of cell and network. Second, we provide the training procedure for cell architecture search. We conclude the subsection with the overview of obtained results and show the learned cells' architectures.

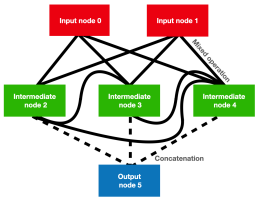

Figure 3. The chosen cell architecture. The red nodes are the input nodes; the green nodes are the intermediate nodes; the blue node is the output node. Solid lines correspond to mixed operations between nodes; dash lines correspond to concatenation of the intermediate nodes' outputs.

Operations. Following (Liu et al., 2018a), we use the further set of operations: separable convolutions with kernel sizes 3×3 (padding 1) and 5×5 (padding 2), dilated separable convolutions with kernel sizes 3×3 (padding 2) and 5×5 (padding 4), max pooling and average pooling with kernel size 3×3 (both with padding 1), identity and zero operation. Padding is used to preserve spatial dimensions of output of each operation. According to (Liu et al., 2019b), we apply separable convolutions twice and enclose convolutions with ReLU and BatchNorm.

Cell. In the previous works (Liu et al., 2019b), (Liu et al., 2018a), the authors typically use two types of cells – normal and reduction cells. While the general architecture of both cells is the same, normal cell preserves spatial dimensions and reduction cell decreases spatial dimensions. In practice, this is achieved by using stride 1 in all operations of a normal cell and stride 2 in operations adjacent to the input nodes of a reduction cell. While we generally follow the architecture of a cell given in (Liu et al., 2019b), we slightly reduce the number the nodes and the overall cell architecture is shown in Figure 3. We simultaneously learn both normal and reduction cells with corresponding α_{normal} and $\alpha_{\text{reduction}}$.

Shape adjustment. There are two issues related to the shapes mismatch. First, the number of output channels in the preceding cell may be incompatible with the number of input channels in the subsequent cell. This issue is usually solved by applying convolutions (Liu et al., 2018a) with kernel size 1×1 and stride 0, accompanied with ReLU and BatchNorm.

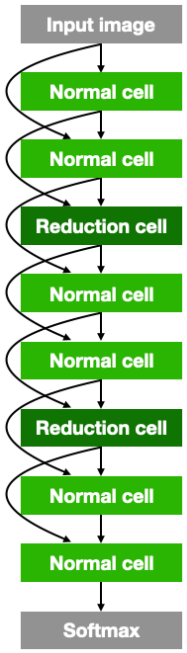

Figure 4. Network for architecture search that is used in cell architecture search.

Another shape mismatch occurs after applying reduction cell: due to decreased spatial dimensioanality, there is a need to adjust spatial shape of another input as well. In this case, two convolutions are applied (Liu et al., 2018a). Each convolution acts on a separate spatial part of the input, and the output number of channels is divided in half between these convolutions. ReLU and BatchNorm are also applied here.

Network. Inspecting the works (Liu et al., 2018a), (Liu et al., 2019b), (Real et al., 2018), we observe that the conventional approach is to repeat blocks of several normal cells and a single reduction cell to form the network for architecture search. Thus, we use similar model, schematically shown in Figure 4. We use 8 layers in total, 2 normal cells alternating with a reduction cell. We also apply additional intermediate layers to force shapes' compatibility. Initial number of channels is set to 16. We add a classifier on top of the last normal cell, see Appendix A. When cell architecture search is finished, we discretize the learned architecture according to method described in (Liu et al., 2019b): for each intermediate node, we choose top-2 operations from distinct preceding nodes (excluding zero operation) with the highest probabilities.

Training details. We largely follow the setup from (Liu et al., 2019b). We set batch size to 64 and train the model for 40 epochs. For model parameters w, we use SGD with initial learning rate 0.025 decreasing down to zero according

Table 2. Evaluation of the learned cells and estimate of computational cost in terms of time per iteration (sec).

	Train accuracy	Validation Accuracy	Time per iteration
$\xi = 0$	99.96%	71.42%	1.07
$\xi > 0$	99.97%	74.64%	2.30

to cosine annealing schedule, momentum 0.9 and weight decay $3e-4$. For parameters $\alpha = (\alpha_{\text{normal}}, \alpha_{\text{reduction}})$ we use Adam optimizer with learning rate $3e-4$, weight decay $1e-3$, $\beta = (0.5, 0.999)$. We have zero initialization for α to provide uniform distribution over operations at the beginning of training. Cell architecture search is performed on the CIFAR10 dataset. Following (Liu et al., 2019b), we split CIFAR10 training dataset into train and validation parts. We run experiments both for first-order approximation, i.e. $\xi = 0$, and second-order approximation, i.e. $\xi > 0$. For the second-order approximation, we follow the proposition from (Liu et al., 2019b) and set ξ to be equal to the learning rate of w's optimizer.

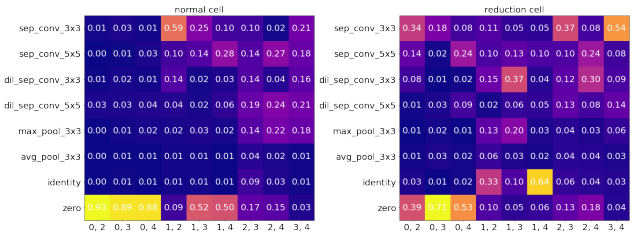

Figure 5. Values of α_{normal}, $\alpha_{\text{reduction}}$ at the end of training for cell architecture search with $\xi = 0$. X-axis corresponds to edges between nodes, Y-axis corresponds to operations.

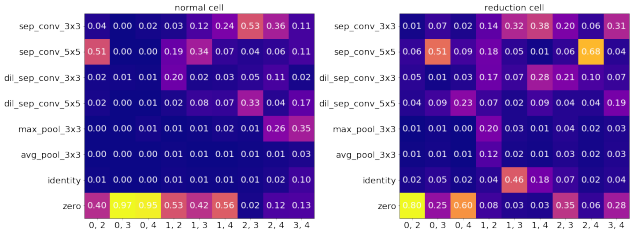

Figure 6. Values of α_{normal}, $\alpha_{\text{reduction}}$ at the end of training for cell architecture search with $\xi > 0$. X-axis corresponds to edges between nodes, Y-axis corresponds to operations.

Results. In Figures 5, 6 we show the values of α_{normal}, α_{reduce} at the end of training (after softmax is applied). For the corresponding normal and reduction cells see Figures 7, 8. As for qualitative differences, we note that pooling operation (more precisely, max pooling) is selected in the final cell architecture only when $\xi > 0$. For quantitative evaluation of the learned cells, we train the network with discretized cells from scratch for 100 epochs using the same training procedure and train/validation split, see Table 2 for the results. We obtain that with second-order approximation, the learned cells achieve better performance: 74.64% validation accuracy in comparison with 71.42% for first-order approximation. At the same, second-order approximation takes almost twice longer time than first-order approximation. These results are consistent with preliminary investigation of bilevel optimization from the toy experiment in subsection 4.1.

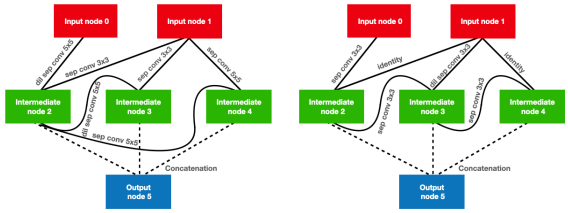

Figure 7. The learned architectures of normal (left) and reduction (right) cells when $\xi = 0$.

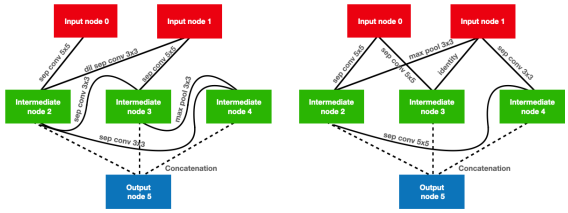

Figure 8. The learned architectures of normal (left) and reduction (right) cells when $\xi > 0$.

4.3. Final Network Training

In this section, we present the experiments of training the final network architecture consisting of the learned normal and reduction cells both for $\xi = 0$ and $\xi > 0$.

Network. Final network architecture consists of 14 cells in total, 4 normal cells alternate with a reduction cell. As previously, the number of initial channels is 16. Similarly, we use auxiliary layers to preserve shapes' compatibility and add linear classifier on top the last normal cell, see Appendix A.

Training details. We train the final network on CIFAR10 from scratch. We keep the training procedure the same as for architecture search except that train for 600 epochs with batch size 128. We do not use additional enhancements such as cutout (DeVries & Taylor., 2017), path dropout and auxiliary towers as it is done in (Liu et al., 2019b).

Results. Table 3 shows the performance of the final networks trained on CIFAR10. While both models are able to achieve 99.9% accuracy on train dataset, cells learned

Table 3. Final network performance trained on CIFAR10 and CIFAR100.

	CIFAR10		CIFAR100	
	Train accuracy	Test accuracy	Train accuracy	Test accuracy
$\xi = 0$	99.96%	78.28%	99.94%	38.45%
$\xi > 0$	99.95%	79.67%	99.94%	46.41%

with second-order approximation allow to obtain better test performance: increase by 1.39%. This result is consistent with outcomes from subsection 4.2.

4.4. Cell transfer

The authors (Liu et al., 2019b) show that the cell architecture learned with DARTS approach on one dataset can be successfully transferred to another dataset. We perform similar experiment, where we transfer the cells learned in subsection 4.3 to the CIFAR100 dataset. While the image domains are quite similar, CIFAR100 contains much more classes and less images per class, thus, constitutes a more complicated task.

Network. For CIFAR100, we use exactly the same final network architecture as in 4.3, see Appendix A.

Training details. We follow the same training procedure as in 4.3, except that the number of training epochs is set to 300 while the batch size is 64.

Results. Table 3 shows the performance of final network trained on CIFAR100 dataset. Similarly, both models are able to achieve 99.9% on train dataset while the performance on test data is worse. The reason may be that the overall network architecture is not expressive enough for this data and/or the training procedure may be not optimal. Nevertheless, we see sufficient increase by 7.96% in test performance when using cells obtained from second-order approximation. This confirms the transferability of the learned cells to more complex problems, similarly to the observation from the paper (Liu et al., 2019b).

4.5. Discussion

As described above, our experimental results confirm the superior performance of the cells learned with second-order approximation over the cells learned with first-order approximation. The reported results are based on a single run of each training procedure. However, as stated in (Liu et al., 2019b), Figure 3, the performance of the learned cells may vary a lot, especially in the beginning of training. Thus, multiple runs are required to to obtain more robust estimate of the learned cell's performance. Also, performance comparison with other approaches for neural architecture search would be highly relevant to better assess the DARTS approach. But these directions imply larger-scale study and we leave it for the future work.

5. Conclusion

In this project, we implement the DARTS approach based on the description from the paper (Liu et al., 2019b) and prior works (Liu et al., 2018a). Our observations are consistent with the experiments' results from the original work (Liu et al., 2019b): second-order approximation allows to obtain more expressive cell architecture than first-order approximation but also is more computationally expensive. Moreover, we observe that this outcome also holds when the cells learned on CIFAR10 are transferred to CIFAR100. Our implementation of DARTS and experiments can be found here.[1]

References

Ahmed, K. and Torresani., L. Connectivity learning in multi-branch networks. 2017.

Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. *ICML*, 2018.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang., J. Efficient architecture search by network transformation. *AAAI*, 2018.

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

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

Chen, X. and Hsieh., C. Stabilizing differentiable architecture search via perturbationbased regularization. *ICML*, 2020.

Chen, X., Xie, L., Wu, J., and Tian., Q. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *ICCV*, 2019.

DeVries, T. and Taylor., G. W. Improved regularization of convolutional neural networks with cutout. 2017.

Dong, X. and Yang., Y. Searching for a robust neural architecture in four gpu hours. *CVPR*, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer,

[1] https://github.com/VoronkovaDasha/project_darts

M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby., N. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Elsken, T., Metzen, J., and Hutter., F. Neural architecture search: A survey. *JMLR*, 2019.

Hsu, W., Bolte, B., Tsai, Y., Lakhotia, K., Salakhutdinov, R., and Mohamed., A. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. 2021.

https://paperswithcode.com/sota/neural-architecture-search-on imagenet.

Jiang, Y., Hu, C., Xiao, T., Zhang, C., and Zhu., J. Improved differentiable architecture search for language modeling and named entity recognition. *EMNLPIJCNLP*, 2019.

Li, G., Qian, G., Delgadillo, I., Muller, M., Thabet, A., and Ghanem., B. Sgas: Sequential greedy architecture search. *CVPR*, 2020.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A., Huang, J., and Murthy, K. Progressive neural architecture search. 2018a.

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

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu., K. Hierarchical representations for efficient architecture search. *ICLR*, 2018b.

Liu, H., Simonyan, K., and Yang., Y. Darts: Differentiable architecture search. *ICLR*, 2019b.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean., J. Efficient neural architecture search via parameter sharing. 2018.

Real, E., Aggarwal, A., Huang, Y., and Le., Q. Regularized evolution for image classifier architecture search. 2018.

Real, E., Aggarwal, A., Huang, Y., and Le., Q. V. Aging evolution for image classifier architecture search. *AAAI*, 2019.

Ren, P., Xiao, Y., Chang, X., Huang, P., Li, Z., Chen, X., and Wang., X. A comprehensive survey of neural architecture search: Challenges and solutions. 2020.

Saxenaa, S. and Verbeek., J. Convolutional neural fabrics. *NeurIPS*, 2016.

Shin, R., Packer, C., and Song., D. Differentiable neural network architecture search. 2019.

Veniat, T. and Denoyer., L. Learning time/memory-efficient deep architectures with budgeted super networks. *CVPR*, 2018.

Wang, D., Gong, C., Li, M., Liu, Q., and Chandra., V. Alphanet: Improved training of supernets with alpha-divergence. *ICML*, 2021.

Wen, W., Liu, H., Chen, Y., Li, H., Bender, G., and Kindermans., P. Neural predictor for neural architecture search. *ECCV*, 2020.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer., K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. 2019.

Xie, S., Zheng, H., Liu, C., and Lin., L. Snas: stochastic neural architecture search. *ICLR*, 2019.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., and Xiong., H. Pc-darts: Partial channel connections for memory-efficient architecture search. *ICLR*, 2020.

Zoph, B. and Le, Q. Neural architecture search with reinforcement learning. 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le., Q. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.

A. Architecture details

In the stage of architecture search, we use network with classifier shown in Table 4 with $w = 2048$ and num. classes equal to 10. In the stage of network training, we use final network architecture shown in Figure 9 with classifier from Table 4 with $w = 4096$ and num. classes equal to 10 for CIFAR10 and num. classes equal to 100 for CIFAR100.

Table 4. Architecture of classifier used during architecture search and final network training.

Classifier
Linear(w), ReLU
Linear(1024), ReLU
Linear(256), ReLU
Linear(num. classes)

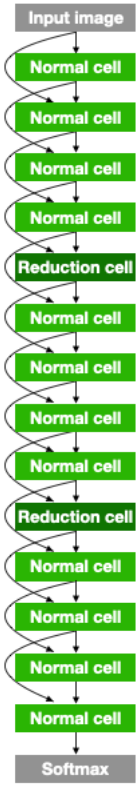

Figure 9. Final neural network architecture.