IMPROVING DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH WITH SPARSE CONNECTIONS AND MODEL PRUNING

GRIGOR BEZIRGANYAN
Technical University of Munich
Master's student
grigor.bezirganyan@tum.de

HAYK AKARMAZYAN
National Polytechnic University of Armenia
PhD in Engineering, Associate Professor
akarmazyan@gmail.com

DOI: 10.54503/2579-2903-2022.1-203

Abstract

Neural networks have contributed to many breakthroughs across several disciplines. Their ease of use and scalability have motivated the development of many techniques in computer vision, natural language processing, audio analysis, etc. The neural network architecture plays a dominant role in its performance, and there have been many advances on designs and strategies for defining efficient neural networks. However, manually tuning neural architectures requires a significant amount of time and expert knowledge. To overcome the difficulty of manually setting up the architecture for a neural network, Neural Architecture Search (NAS) has gained popularity. NAS methods involve three general dimensions, namely search space, search strategies, and performance estimation strategies **[1].** Different approaches vary in these dimensions.

DARTS **[2]** corresponds to a cornerstone work for differentiable NAS. It is an order of magnitude faster than evolutionary and reinforcement learning-based algorithms and produces intuitive results which have shown a good performance on several computer vision and natural language processing tasks. Still, DARTS has several pitfalls which have motivated this work. Namely, the method suffers from instability to hyperparameters, as shown in NAS-Bench-201 experiments **[4].** Moreover, it can quickly lead to degenerate neural architectures, which do not perform well at all, in case the structure of the network is not configured properly in the search phase. DARTS also depends on several humandesigned heuristics, which can affect the final performance significantly if not selected properly. Finally, the heuristics for building the final architecture lead to a big discrepancy from the architecture used in the search phase, which leads to instability between the search phase and the final network performance.

In this work, we focus on tackling the drawbacks of DARTS, by incorporating in the search phase a gradual pruning strategy while using single-level optimization, promoting the network connections to be sparse. Careful analysis of the weights shows that our method achieves more stable results, is less prone to degenerate architectures and indeed leads to better ones, and does not require heuristics to get the final neural architecture. The source code is available on demand, to facilitate reproducibility.

Keywords and phrases: Neural Architecture Search, DARTS, Sparse Models, Deep Learning

ՆԵՅՐՈՆԱՅԻՆ ՃԱՐՏԱՐԱՊԵՏՈՒԹՅԱՆ ԴԻՖԵՐԵՆՑՎՈՂ ՈՐՈՆՄԱՆ ԲԱՐԵԼԱՎՈՒՄԸ ՆՈՍՐ ՄԻԱՑՈՒՄՆԵՐԻ ԵՎ ՄՈԴԵԼԻ ԷՏՄԱՆ ՄԻՋՈՑՈՎ

ԳՐԻԳՈՐ ԲԵԶԻՐԳԱՆՅԱՆ
Մյունխենի Տեխնիկական Համալսարան
մագիստրոս
grigor.bezirganyan@tum.de

ՀԱՅԿ ԱԿԱՐՄԱԶՅԱՆ
Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան
ճարտարագիտության թեկնածու, դոցենտ
akarmazyan@gmail.com

Համառոտագիր

Նեյրոնային ցանցերը նպաստել են գիտական մի շարք ոլորտներում տարբեր բացահայտումների: Դրանց օգտագործման հեշտությունը եւ մասշտաբայնությունը խթանել են համակարգչային տեսողության, բնական լեզվի մշակման, աուդիո վերլուծության եւ այլ բազմաթիվ տեխնիկաների մշակմանը: Նեյրոնային ցանցի ճարտարապետությունը առանցքային դեր է խաղում դրա արդյունավետության մեջ, այդ պատճառով բազմաթիվ առաջխաղացումներ են գրանցվել դրանց նախագծման մեջ՝ ավելի արդյունավետ ցանցեր ստանալու համար: Սակայն, նեյրոնային ճարտարապետության ձեռքով կարգավորումը պահանջում է զգալի քանակությամբ ժամանակ եւ փորձագիտական գիտելիքներ: Նեյրոնային ցանցի ճարտարապետությունը ձեռքով կարգավորելու դժվարությունը հաղթահարելու համար մեծ տարածում է գտել նյարդային ճարտարապետության որոնումը (NAS): NAS մեթոդները ներառում են երեք ընդհանուր չափումներ՝ որոնման տարածություն, որոնման ռազմավարություններ եւ որակի գնահատման ռազմավարություններ **[1]:**

DARTS-ը **[2]**, լինելով դիֆերենցվող ճարտարապետության որոնման եղանակ, կարեւոր նշանակություն է ունեցել NAS-ի մեջ: Այն մի քանի անգամ արագ է, քան էվոլյուցիոն եւ ամրապնդման ուսուցման վրա հիմնված ալգորիթմները եւ տալիս է ինտուիտիվ արդյունքներ, որոնք լավ արդյունք են տալիս համակարգչային տեսողության եւ բնական լեզվի մշակման մի շարք խնդիրներում: Այնուամենայնիվ, DARTS-ն ունի մի քանի թույլ կողմեր, որոնք դրդել են այս աշխատանքը գրելուն: Մասնավորապես, մեթոդում առկա է հիպերպարամետրերի անկայունության խնդիրը, ինչպես ցույց է տրված NAS-Bench-201 փորձերում **[4]:** Ավելին, դա կարող է արագ հանգեցնել նեյրոնային ճարտարապետության դեգեներացմա: Դրանք բոլորովին լավ չեն աշխատում, եթե որոնման փուլում ցանցի կառուցվածքը ճիշտ չի կազմված: DARTS-ը նաեւ կախված է մարդու կողմից նախագծված մի քանի

էվրիստիկներից, որոնք կարող են էապես ազդել վերջնական արդյունքի վրա, ճիշտ չընտրվելու դեպքում: Վերջապես, վերջնական ճարտարապետության կառուցման էվրիստիկաները հանգեցնում են որոնման փուլում օգտագործվող ճարտարապետության մեծ անհամապատասխանության, ինչը հանգեցնում է որոնման փուլի եւ ցանցի վերջնական արդյունքի միջեւ անկայունության:

Այս աշխատանքում մենք կենտրոնանում ենք DARTS-ի թերությունների վերացման վրա՝ որոնման փուլում ներառելով աստիճանական էտման ռազմավարություն՝ միաժամանակ օգտագործելով մեկ մակարդակի օպտիմիզացում՝ նպաստելով ցանցային կապերի նոսրացմանը: Կշիռների մանրակրկիտ վերլուծությունը ցույց է տալիս, որ մեր մեթոդը գրանցում է ավելի կայուն արդյունքներ, ավելի քիչ հակված է դեգեներացված ճարտարապետություններին, եւ չի պահանջում էվրիստիկա՝ վերջնական նեյրոնային ճարտարապետությունը ստանալու համար: Աղբյուրի կոդը հասանելի է պահանջի՝ վերարտադրելիությունը հեշտացնելու համար:

Բանալի բառեր և բառակապակցություններ. նեյրոնային ճարտարապետության որոնում, DARTS, նոսր մոդելներ, խորը ուսուցում:

УЛУЧШЕНИЕ ПОИСКА ДИФФЕРЕНЦИАЛЬНОЙ НЕЙРОННОЙ АРХИТЕКТУРЫ С РАЗРЕЖЕННЫМИ СОЕДИНЕНИЯМИ И ОБРЕЗАНИЕМ МОДЕЛИ

ГРИГОР БЕЗИРГАНЯН
Технический университет Мюнхена
магистрант
grigor.bezirganyan@tum.de

АЙК АКАРМАЗЯН
Национальный политехнический университет Армении
кандидат технических наук, доцент
akarmazyan@gmail.com

Аннотация

Нейронные сети сыграли большую роль в совершении открытий во многих областях науки. Их простота использования и масштабируемость послужили стимулом для разработки многих методик в сфере компьютерного зрения, обработки естественного языка, анализа звука и других. Архитектура нейронной сети имеет приоритетное значение в ее производительности, а в области разработки и стратегий определения эффективных нейронных сетей было уже достигнуто немало результатов. Однако ручная настройка нейронных архитектур требует значительного количества времени и экспертных знаний. В деле преодоления трудности ручной настройки архитектуры нейронной сети популярность приобрела система поиска нейронной архитектуры (NAS). Методы NAS включают в себя три основных измерения, а именно: пространство поиска, стратегии поиска и стратегии оценки производительности **[1].** Различные подходы отличаются в соответствии с этими измерениями.

DARTS **[2]** представляется краеугольным камнем работы по дифференцируемой NAS. Он на порядок быстрее алгоритмов, основанных на эволюционном обучении и обучении с подкреплением, и дает интуитивно понятные результаты, которые показали высокую производительность в разных задачах компьютерного зрения и обработки естественного языка. Тем не менее у DARTS есть несколько недостатков, которые стали отправной точкой для этой работы. А именно: метод страдает неустойчивостью к гиперпараметрам, как показано в экспериментах NAS-Bench-201 **[4].** Более того, это может быстро привести к вырождению нейронных архитектур, которые вообще плохо работают, если структура сети не настроена должным образом на этапе поиска. DARTS также зависит от нескольких разработанных человеком эвристик, которые могут значительно повлиять на конечную производительность, если они не подобраны должным образом. Наконец, эвристики построения окончательной архитектуры приводят к большому расхождению с архитектурой, используемой на этапе поиска. Это приводит к нестабильности между этапом поиска и конечной производительностью сети.

В данной работе мы сосредоточимся на устранении недостатков DARTS посредством добавления стратегии постепенного сокращения на этапе поиска, будем использовать одноуровневую оптимизацию, способствуя разреженности сетевых подключений. Тщательный анализ весового значения показывает, что

наш метод дает более стабильные результаты, менее склонен к вырождению архитектур и способствует их более высокой эффективности, а также не требует эвристики для получения окончательной нейронной архитектуры. Для облегчения воспроизводимости исходный код доступен по запросу.

Ключевые слова и словосочетания: поиск нейронной архитектуры, DARTS, разреженные модели, глубокое обучение

Introduction and Related Work

Gradient-based Neural Architecture Search (NAS) has become very popular in the last several years with the introduction of DARTS **[2].** This is a fast search procedure compared to other NAS methods, showing good results on different benchmarks **[5].**

In the setting of differentiable NAS, the aim is usually to learn a good structure for a cell that should be reused across wider/deeper networks. One of the fundamental ideas of gradient-based NAS was proposed in DARTS **[2],** where the network learns both the weights and the connections of the layers at the same time. Figure 1 demonstrates the representation of one cell in the neural architecture that DARTS can produce. Initially, the connections between the nodes are unknown (a) and all the possible operations are placed as choices on the edges (b). Afterwards, the weights of the network are learned along with the weights of the connections between nodes (c). Eventually, only the operations with the highest connectivity weights are selected (d). It is important to note that each node within the cell is connected to all its predecessors. So, node 4 is connected to all the nodes from 1 to 3, node 3 to nodes 1 and 2, and node 2 is connected to node 1. Each connection weight between the nodes is characterized with value $a_{i,j}$, which is passed through a softmax() function before reaching the next node.

In the original DARTS settings, the possible operations are defined as sep_conv_3 × 3, sep_conv_5 × 5, dil_conv_3 × 3, dil_conv_5 × 5 , avg_pool_3 × 3, max_pool_3 × 3, skip_connect, and none. The numbers a × b represent the kernel size, sep_conv represents a separable convolution, dil_conv is a dilated convolution with dilation rate 2, avg_pool and max_pool are the average pooling and max pooling operations, correspondingly, skip_connect propagates the input further without modifying it, and a none connection just multiplies the input by 0. The network weights are optimized on the training set, while the connection weights are optimized on the validation set. For each epoch, the model first updates the connection weights on the validation set and afterward tunes the weights on the training set. The target is defined by bilevel optimization, where the aim is to minimize the validation loss by choosing the best architecture weights α. The authors argue that single-level optimization will have problems because the α weights can overfit on the training data. Formally, the bilevel optimization procedure corresponds to:

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha),$$
$$\text{s.t. } w^*(\alpha) =_w \mathcal{L}_{train}(w, \alpha).$$

As the inner optimization of $_w\mathcal{L}_{train}(w, \alpha)$ can be very expensive, especially for each new update on α (for each α the loss $\mathcal{L}_{train}$ needs to be calculated until convergence), it would be infeasible to compute the whole $w^*(\alpha)$ each time α is

updated. The authors propose an approximation for $w^*(\alpha)$, in which the whole inner optimization is approximated by one inner loop on the training set, and the ∇_w is multiplied by a chosen ϵ value, which is a user-defined hyperparameter. Therefore,

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx$$
$$\nabla_\alpha \mathcal{L}_{val}(w - \epsilon \nabla_w \mathcal{L}_{train}(w, \alpha)), \alpha).$$

Using the finite difference approximation, this is further expanded into:

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx$$
$$\nabla_\alpha \mathcal{L}_{val}(w', \alpha) -$$
$$\epsilon \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\varepsilon}.$$

This approximation leads to a big speedup, reducing the complexity from $O(|\alpha| \cdot |w|)$ to $O(|\alpha| + |w|)$. If $\epsilon = 0$, the second part of the equation is completely discarded, which corresponds to assuming that the current w is the optimal one. From the experiments, the authors have seen that this choice can contribute to a speedup, but worse performance. Therefore, the ϵ hyperparameter was chosen to be > 0 in the experiments.

Even though DARTS performed well on several benchmarks, it has several flaws which have been identified by subsequent studies **[6–8].**

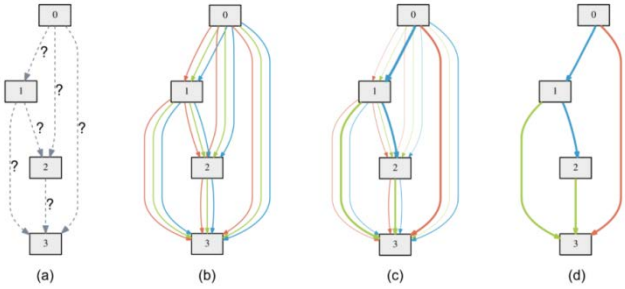

Figure 1.

Improving DARTS in the Selection of the Final Architecture

Wang et al **[9]** claimed that the magnitude of the architecture weights α does not necessarily indicate importance in the final architecture. The authors proposed to use a perturbation-based method to select the final architecture. Instead of selecting the top connections as done in DARTS, the authors train the super-network with all the operations until the end (exactly like DARTS), after which they randomly select a pair of nodes and determine the best connection for them by removing an operation and tracking the performance drop on the validation set. After the operation is selected, the network is finetuned for 5 epochs, after which another random node pair is selected, and the procedure for selecting the best operation is repeated. This entire process repeats until all the node pairs are processed. Bi et al. **[10]** identified that DARTS, and also extensions such as PC-DARTS **[3]** or P-DARTS **[16]** which we describe later, suffer from instability. Several trials of the same experiment lead to very different final architectures, also with a very different final performance. The authors proposed to

bridge the gap between the super-network training and the final sub-network selection with an amended hessian trick, which approximates the second level derivative for the architecture parameters more accurately. They claim that the approximation in the bi-level optimization in DARTS causes big instabilities and accumulates errors through time, and the amended hessian is shown to perform better and lead to more stable architectures. The authors also unify the hyperparameters used in the search and final full training procedure, to further bridge the gap between the two phases. With the complete approach, the network reaches stability and does not have the none connections dominating the final cell structure. Liu et al. **[2]** have chosen to ignore the none connections in the original DARTS algorithm, but that issue is avoided by applying the amended gradient estimation.

Improving DARTS Through the Use of Single-Level Optimization

Hou and Jin **[11]** demonstrated that bi-level optimization and the separate update of architecture parameters and weights causes the non-learnable parameters to take over the learnable ones. This causes most of the final architectures to consist of either none connections or skip_connections. The authors re-evaluated the decision of considering bi-level optimization in DARTS, reverting back to a single-level optimization procedure. They also point out that the learning rate of the architecture parameters plays a big role in the final architecture, and a large learning rate can lead to architectures that feature many non-learnable parameters. Finally, the authors used a sigmoid() activation function instead of a softmax() to diminish the bias caused by non-learnable parameters. Experiments on the NAS-Bench-201 benchmark **[4]** demonstrate that this approach can lead to very promising results.

To optimize the search time, the algorithm stops as soon as the size of the model (i.e., the number of parameters) drops below the threshold of the expected model size. Li et al **[12]** proposed to use mirror descent with a single-level objective, instead of a bilevel objective for a neural architecture search. The authors claim that it is more beneficial to treat the architecture weights as learned parameters instead of hyperparameters. Empirical results show that this approach outperformed the state-of-the-art solutions at the time, on the DARTS search space.

In our experiments, we have used single-level optimization as well. Yet, unlike other studies where the authors used sigmoid() or softmax() activation functions for the architecture weights, we have used the recently proposed entmax() function to promote sparsity in the connections.

Network Pruning and Other Model Adjustment Strategies for DARTS

The main flaw in DARTS pointed out by Chen et al **[6]** is the gap between the search phase and the final evaluation phase. During the search phase, a smaller architecture is used with a smaller number of layers, while for the final evaluation, the number of layers/cells is increased drastically. That causes a big discrepancy between the search and final evaluation procedures.

In P-DARTS, Chen et al **[6]** proposed to gradually increase the number of cells in the search procedure, to gradually match the final evaluation scenario. DARTS **[2]** uses 8 cells for training and 20 cells for the final evaluation. P-DARTS proposed to gradually increase the number of cells, starting from 5, then setting the number to 11, then to 17, and finally doing the evaluation on a 20 cell network.

In their experiments, Chen et al **[6]** also noticed that with the increase of the depth of the network, the number of skip-connections dominates the architecture. To overcome that, they add dropout on top of the skip_connect operation, and decrease the

dropout probability gradually. Moreover, the authors constrained the number of skip-connections to a fixed number M in the final architecture. The value of M is another hyperparameter tuned by the user. Laube and Zell **[13]** proposed to prune bad candidates in a DARTS super-network, replacing them with new more promising ones. The authors claim that learnable operations are more difficult to tune, and the network needs to be trained for several epochs with frozen architecture weights so that the learnable parameters become competent with non-learnable ones. To expand the architecture, the authors introduce morphisms for all the operations used in the connections:

- For convolutions, expanding the kernel size by initially padding the kernel with zeros;
- Increase the channel count of convolutions by initially setting the weights to zeros;
- Inserting new layers with nonlinearities;
- Decreasing the kernel size by keeping only the central part;
- Increasing or decreasing the dilation size in dilated convolutions.

For each morphing stage, they artificially keep at least one convolution, in order to have learnable operations. The whole procedure takes 100 epochs to finish and demonstrates good results in terms of the speed of convergence and the number of new operations added to the search space. One of the drawbacks of DARTS noted by Noy et al. **[14],** is the discrete procedure of extracting the final architecture from the architecture weights. With ASAP, Noy et al **[14]** proposed to extract the final architecture in a continuous manner. The authors suggest using a combination of an annealing schedule and a pruning policy, which guarantees that the pruning will not affect the quality of the final cell. To allow a fair competition for the learnable parameters, they also fix the architecture weights for several epochs at the beginning. When doing the pruning of the network, the connections are removed if their weights become lower than a threshold. The threshold T at time t follows a designed schedule with annealing. The search procedure stops when all the node pairs have only a single connection operation.

Other DARTS Improvements

DARTS+ **[15]** demonstrated that DARTS easily falls into a performance collapse, in which all the learnable connections get dominated by non-learnable operations when trained for long enough. The authors claim that the collapse happens because of overfitting in the bi-level optimization of the DARTS architecture weights. DARTS+ uses two early stopping strategies to prevent performance collapse. The first technique states that the search procedure needs to be stopped as soon as there exists a node pair for which the operations do not change their order for N epochs. The second method stops the training as soon as there are two or more skip-connections in a normal cell. Chu et al **[16]** identified that skip-connections in DARTS have an unfair advantage over all the other operations. The authors proposed to eliminate that advantage by adding an auxiliary skip connection from the input directly to the output. The weight of the auxiliary connection decays as the search procedure approaches the end. The same authors also proposed using a sigmoid() activation function to enforce collaboration between operations, instead of competition through softmax(). Yet, these changes alone do not solve the problems of DARTS, as there is a big gap between the search phase and the final architecture selected for the full training phase. To be able to

select the final architecture, the authors also change the loss function to a zero-one loss, to further stimulate the architecture weights to take values of either 0 or 1. To reduce the memory footprint Xu et al **[3]** proposed to use channel regularisation, where the model selects only the top 1 /K channels from the convolutional feature maps to reduce the memory footprint K times. The selected feature maps are used for node computations, while the others are propagated directly to the output of the node. That resulted in a lower memory footprint and thus a bigger batch size could be used improving the stability of the architecture search. As on each iteration, some random set of channels is selected, which can affect the optimal connectivity determined by them and make it unstable. To address that issue the authors propose another weighting of softmaxi ($\beta_{i,j}$) for each node i that gets inputs from nodes j. That β weight is aimed to weigh the inputs and add more stability to the optimal connectivity between nodes. These optimizations helped to perform a neural architecture search on a single GTX 1080 Ti in only 0.1 GPU-days for CIFAR10 and 3.8 GPU-days for ImageNet datasets.

An Experimental Analysis on DARTS

Figure 2.

This section presents a detailed experimental analysis on the limitations of DARTS. The analysis was conducted with standard datasets in the area, using the search spaces from the original DARTS publication **[2],** and also from NAS-Bench-201 **[4].**

Experimental Setup

As the NAS problem implies a search for an architecture, there are two phases in all the experiments. The first is the search phase, where the aim is to find a good architecture. The second is the full training phase, where the aim is to evaluate the architecture. The final full training phase uses the default train/test split of the dataset, while the search phase uses only the train split, so that the approach does not overfit the validation set. There are two search spaces on which we evaluate and report the results. The DARTS search space contains 8 different operation types, and the NAS-Bench-201 search space features 5 operation types. To fit the NAS-Bench-201 **[4]** settings, we use 17 layers instead of 8 during the search phase. All the experiments were conducted with a fixed hyperparameter set following Liu et al **[2].** To be comparable to DARTS, we use the default hyperparameters for all the aspects, except the sparsity and pruning of the network. For the NAS-Bench-201 experiments, we also change the depth of the network to fit the experimental setup of NAS-Bench-201 **[4].**

As some experiments incorporate pruning, the search procedure for those is drastically accelerated, opening space for searching for more epochs in the same amount of time. We stopped the search procedure at approximately the same wall-clock time, which was required to complete the search procedure in standard DARTS settings, setting the number of epochs to 200.

In our experimental setup, we used popular datasets in the field of NAS. More specifically, we have concentrated on the CIFAR-10 **[17]** and CIFAR-100 **[18]** datasets. Both of these are computer vision benchmarks containing images from different classes, where models are tasked with classifying each image into a corresponding class. The CIFAR-10 dataset consists of 60000 colored images with a size of 32 × 32. The training split consists of 50000 images, while the test split consists of 10000. There are 10 classes in the dataset: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. All the classes are mutually exclusive. The CIFAR-100 dataset is similar to CIFAR-10, but contains 100 different classes. For each class, there are 500 training images and 100 testing ones, colored and with a size of 32 × 32. We did not consider experiments on the ImageNet **[19]** dataset, due to the large amount of compute needed to process it. Even though some approaches for NAS have used the ImageNet dataset, these studies have also reported their results on CIFAR-10 and CIFAR-100, and thus comparing this work with others is possible.

Analysis of the DARTS Search Procedure

Figure 3.

In the DARTS search phase, the normal cell is the main contributor to the final performance of the model. Naturally, most prior work has focused on fixing the issues in the normal cell **[8, 11, 15, 16, 20]**, and we also concentrate on the analysis of the normal cell in vanilla DARTS. Figure 2 illustrates the change of architecture weights throughout the training epochs, in an initial experiment with vanilla DARTS over CIFAR-100. The weights are plotted after applying the softmax () activation on the weights, as the values are used after the softmax() when multiplied by their corresponding operations. When analyzing the final configuration of the architecture weights, the none connections are the dominant ones across all the node connections. For some node pairs, the weights of other connections are so small that the none connection contributes to most of the output of that node.

Liu et al **[2]** have simply chosen to ignore the none connections as a whole, when determining the final architecture. The DARTS algorithm picks the best connection per node pair after ignoring the none connections, and keeps only 2 connections per node (i.e. node 2 gets 2 inputs, node 3 gets 2 inputs, node 4 gets 2 inputs, and node 5 gets 2 inputs). This results in an unfair architecture choice, as most

of the connections have a very small impact on the output of a node over the last few epochs of the search phase. In the example of Figure 2, none of the connections for the node pair $4 \rightarrow 5$, for instance, is better than others to be considered for the final architecture. Yet, DARTS considers the top connection out of those as an operation in the final architecture, despite the fact that all of them are very close to each other and have a weight of almost 0. In sum, numerically picking the top connection while ignoring the none operation can lead to other operations being chosen while they are not significantly better in terms of their weights.

Analysis of the Super-Network Structure

The performance of the architecture search also depends on the architecture of the super network at the search phase. In the original DARTS search space, the super network consists of 3 blocks of two normal cells, followed by a reduction cell, which is followed by a softmax() normalization layer. In total there are 8 cells (i.e. 6 normal cells and 2 reduction ones). In the NAS- Bench-201 settings, the network is deeper and has 3 blocks of 5 normal cells, followed by a reduction cell and summing up to 17 cells in total. The depth of the super network plays a crucial role in the importance of operations. Several studies **[15, 16]** have demonstrated that deep super-networks have higher chances of having more skip-connections in the final architecture. For deep networks, it is more important to pass the extracted information from the initial layers to the final ones. This promotes bigger weights for skip connections compared to other operations. It should also be noted that, in the standard DARTS procedure, the super-network keeps all the operations up until the very end of the search procedure. Therefore, the operations keep contributing to the output of the cell, despite not having a large weight to be considered in the final architecture. Therefore, the whole final architecture, when ignoring the none connections, often turns into using skip connections for all cases, as shown in the example from Figure 4.

In sum, deep networks can prioritize skip connections over other operations when the super network is deep enough. This can lead to degenerate final architectures, featuring only skip-connections in the normal cells.

Heuristics to Extract the Final Neural Network Architecture

There are several heuristic rules in the DARTS search procedure, for instance for extracting the final architecture from the super-network. Most of the rules work well, even though they contradict the idea of fully automated NAS. One of these rules is the number of nodes in a cell, which is selected to be 4 in DARTS (+ 2 nodes from the previous two cells). Another example is the rule that every node is connected to all its previous ones (e.g., node 4 is connected to nodes 0, 1, 2, and 3). It was assumed that the none connection should remove the unnecessary connections, but the none connection was ignored in the vanilla DARTS **[2].** Lastly, the discretization procedure, where the final architecture is extracted from the super-network, follows the rule of 2 connections per node. In the original DARTS settings, each node in the cell has only 2 inbound connections, and thus node 4 is initially connected to nodes 0, 1, 2, and 3, but in the final architecture it is only connected to two of them. Unlike the first two rules, the final one is a hyperparameter that needs to be tuned by the end-user.

Summarizing the Pitfalls of DARTS

With basis on the aforementioned experiments and the prior work in the area, it can be concluded that there are several issues in the DARTS procedure which affect its performance:

- The architecture weights for none connections overwhelm the rest of the architecture, while the remaining operations do not have large enough margins to have a numerically significant advantage over others.
- Deep super-network architectures collapse to having all skip_connect operations.
- There are many handcrafted heuristics built into DARTS, for instance used to extract the final architecture after the search phase.
- There is a big gap between the search process and the final architecture

Differentiable Neural Architecture Search with Sparsity and Pruning

By analyzing the pitfalls of DARTS, we propose several methods for fixing each of them.

- To avoid model collapse to having all non-learnable none connections, we propose to use a single-level optimization procedure.
- To avoid the collapse to a degenerate network having all skip_connect operations, and also to bridge the gap between the super-network used in the search phase and the final architecture, we propose to prune the operations which do not contribute to the final output.
- We use α-entmax **[21]** to promote sparsity in the architecture weights, instead of the standard softmax() activation.
- To avoid having the constraint of only two connections per node, we select all the inbound connections for the nodes that end up with a weight different from zero.

Figure 5.

Through α-entmax **[21],** small weights are more prone to becoming zeros, and bigger ones grow faster. This effect was used to create sparse architecture weights. As the aim is to pick at most one final operation per node pair, α-entmax can be used to highlight and accelerate the process of picking the top connections. Unlike other studies where the authors pick a threshold to remove a connection **[17,14],** we get rid of the heuristic choice and remove only the connections which become 0 after the α-entmax activation is applied. This also speeds up the search procedure considerably. Formally, α-entmax **[21]** corresponds to computing:

$$\alpha\text{-}entmax(z)_{p\in\triangle^d} p^\top z + H_\alpha(p),$$

where α is the sparsity parameter, Δ^d represents the probability distribution over d possible choices, z is the input feature vector, and $H_\alpha(p)$ is represented by:

$$H_\alpha(p)\begin{cases}\frac{1}{\alpha(\alpha-1)}\sum_j (p_j - p_j^\alpha), & \text{if } \alpha \neq 1\\ -\sum_j p_j \log p_j, & \text{if } \alpha = 1,\end{cases}$$

where the case of α = 1 is the Gibbs-Boltzmann-Shannon entropy. For values of α that are larger than one, $H_\alpha(p)$ results in a sparser probability vector. Figure 5 demonstrates the α-entmax() function for several values of α. The case α = 1 corresponds to the softmax() function, which is smooth, while for bigger α values the function becomes more rigid and a larger span of values is turned into either 0 or 1. Despite the interest in using α-entmax, initial tests replacing the softmax() activation showed that similarly to vanilla DARTS, most of the connections still got dominated by none connections, while the rest are just skip_connections. Assuming that the bi-level optimization favours the none connection and the skip_connect, as pointed out by Dong and Yang **[4],** we combined the use of α-entmax with the use of single-level optimization. Moreover, during the search phase, we prune connections whose weights become zero, removing them from subsequent optimization steps. We analyzed the performance of single-level optimization on the DARTS search space, with α-entmax activation function with the sparsity parameter set to 3, and pruning weights when they decline to 0. Figure 6 presents the evolution of the architecture weights throughout the search epochs. The final architecture extracted from this figure has both skip_connect, none, max_pool, and sep_conv operations. Pruning the weights which have a small impact on the final output of a node helps in minimizing the gap between the search phrase and the full-training phase. Our tests showed that, as in the original DARTS, the final architecture has at most a single connection per node pair, and removing the low scoring operations helps in bridging the gap between the final architecture and the intermediate architecture obtained in the search phase. The other operations, with weights above zero, do not get diminished by an overwhelming number of less important operations contributing to the final output. When using single- level optimization, the none connections do not dominate the whole node pair operation choice, letting other operations demonstrate their usefulness. Note that the final architecture is not degenerate, compared to the one present in bi-level optimization case (i.e., compare Figures 4 and 9). The final architecture also has more clear dominant connections, compared to the ones present when using softmax with single-level optimization. In sum, through the proposed ideas, the search procedure is more stable and results in a well distinguishable dominant operation per node pair (e.g., experiments with different random seeds have resulted in a similar final architecture). Empirical results obtained from these architectures are demonstrated in the next section to validate the proposed ideas.

Figure 6.

Experimental Results

Most of our experiments were conducted in the DARTS search space with the experimental setup following Liu et al **[2],** although we also performed some tests over the NAS-Bench-201 settings **[4].** To understand the contribution of each new idea, we present the results separately for single-level optimization, for α-entmax with pruning, and for the combination of α-entmax with pruning and single-level optimization. Table 1 summarizes the results obtained on both the DARTS and NAS-Bench-201 search spaces. Both the CIFAR-10 and CIFAR-100 datasets were used to compare the performance of different approaches. The table also includes results reported in other studies **[2, 3, 6–8, 13, 16].** Moreover, we have re-run the experiments for the vanilla DARTS, to report the results on both benchmark datasets and the two search spaces. In our results, the bi-level optimization outperforms the single-level optimization. We can also see that α-entmax with sparsity set to 2 does not lead to good performance, but the α-entmax with sparsity 3 and pruning, together with bi-level optimization, outperforms vanilla DARTS. We suspect that 2-entmax worse performance is due to many redundant connections still remaining in the cell during the search phase, and therefore overwhelming the important connections. Using single-level optimization with α-entmax for sparsity and pruning outperforms all the other techniques on the DARTS search space, implying that the proposed technique actually contributes to better results. The experiments on NAS-Bench-201 demonstrate that our approach does not converge to having all skip_connections and performs well on both the CIFAR-10 and CIFAR-100 datasets. The results of DARTS for the NAS-Bench-201 search space reported by Dong and Yang **[4]** have either all skip_connections or none operations as they keep all the top connections obtained in the search process. In sum, the proposed method of applying sparsity to the network and pruning the weights that do not contribute to the final output while also using a single-level optimization shows to perform quite well on both DARTS and NAS-Bench-201 search spaces. Experiments on both CIFAR-10 and CIFAR-100 demonstrate that our method can outperform most of the methods while using the same setup as in vanilla DARTS, without any hyperparameter tuning.

Conclusions and Future Work

Differentiable neural architecture search is a promising approach to learn complex neural architectures without requiring an enormous amount of computational power. We performed an in-depth analysis of the original DARTS method, which helped in understanding how each part contributes to the final output and how to improve several of its aspects. We also proposed a method which uses single-level optimization with the pruning of weak connections, together with α-entmax activations for promoting sparsity in the architecture weights. The experimental results demonstrate the usefulness of the presented ideas. Despite the interesting results, several aspects should be further analyzed in the future. One is to assess the performance of the proposed method on other search spaces, including recurrent cell structures and transformer networks. Instead of designing transformer cells by hand, several options of automating that task through NAS can be explored. Another area that would be interesting to investigate is the effect of initialization of the network weights on the final architecture, and if the initialization could be improved. We have done several experiments by initializing the DARTS weights using unsupervised methods. The preliminary results were not very useful, but we have not pursued them in much detail, therefore some improvement can be made to the speed and quality of DARTS through initialization. One idea that would be interesting to experiment with is to try progressively increasing the sparsity of the network. In our experiments we have fixed the sparsity parameter of the α-entmax, but starting from a dense network and progressively increasing the sparsity enforcement can benefit the final result.

	DARTS		NAS-Bench-201	
Model	CIFAR-10	CIFAR-100	CIFAR-10	CIFAR-100
DARTS† [11,13]	97.24	-	54.30‡	15.61‡
DARTS-† [26]	97.37	82.49	93.80	71.53
PC-DARTS† [12]	97.43	-	-	-
P-DARTS† [16]	97.50	84.08	-	-
PR-DARTS‡ [23]	97.26	82.63	-	-
GOLD-NAS‡ [18]	97.23	-	-	-
MiLeNAS† [17]	97.49	-	-	-
Single-level DARTS‡ [21]	-	-	94.36	73.51
GAEA† [22]	97.50	-	94.10	72.60
ASAP‡ [24]	98.32	-	-	-
DARTS+PT‡ [19]	97.39	-	-	-
Vanilla DARTS† (re-run)	97.24	80.85	97.34	81.40
Single-level DARTS†	95.99	79.80	-	-
Single-level DARTS3†	-	-	-	-
Single-level DARTS4†	-	-	-	-
2-entmax, prune†	97.24	73.01	-	-
3-entmax, prune†	97.35	82.57	-	-
2-entmax, prune, single-level‡	97.42	82.64	-	-
3-entmax, prune, single-level‡	97.58	82.97	97.68	81.75

Table 1.

References

1. Zhu, H.; Zhang, H.; Jin, Y. From federated learning to federated neural architecture search: A survey. Complex & Intelligent Systems 2021, 7, 639–657.
2. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable architecture search. arXiv preprint arXiv:1806.09055 2018.
3. Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.J.; Tian, Q.; Xiong, H. PC-DARTS: Partial channel connections for memory-efficient architecture search. arXiv preprint arXiv:1907.05737 2019. Appl. Sci. 2022, 1, 0 16 of 16
4. Dong, X.; Yang, Y. NAS-Bench-201: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326 2020.
5. Ren, P.; Xiao, Y.; Chang, X.; Huang, P.Y.; Li, Z.; Chen, X.; Wang, X. A comprehensive survey of neural architecture search: Challenges and solutions. ACM Computing Surveys 2021, 54, 1–34.
6. Chen, X.; Xie, L.; Wu, J.; Tian, Q. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In Proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019.
7. He, C.; Ye, H.; Shen, L.; Zhang, T. Milenas: Efficient neural architecture search via mixed-level reformulation. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.
8. Bi, K.; Xie, L.; Chen, X.; Wei, L.; Tian, Q. GOLD-NAS: Gradual, one-level, differentiable. arXiv preprint arXiv:2007.03331 2020.
9. Wang, R.; Cheng, M.; Chen, X.; Tang, X.; Hsieh, C.J. Rethinking architecture selection in differentiable nas. arXiv preprint arXiv:2108.04392 2021.
10. Bi, K.; Hu, C.; Xie, L.; Chen, X.; Wei, L.; Tian, Q. Stabilizing DARTS with amended gradient estimation on architectural parameters. arXiv preprint arXiv:1910.11831 2019.
11. Hou, P.; Jin, Y. Single-level Optimization For Differential Architecture Search. arXiv preprint arXiv:2012.11337 2020.
12. Li, L.; Khodak, M.; Balcan, M.F.; Talwalkar, A. Geometry-aware gradient algorithms for neural architecture search. arXiv preprint arXiv:2004.07802 2020.
13. Laube, K.A.; Zell, A. Prune and Replace NAS. In Proceedings of the Proceedings of the IEEE International Conference On Machine Learning And Applications, 2019.
14. Noy, A.; Nayman, N.; Ridnik, T.; Zamir, N.; Doveh, S.; Friedman, I.; Giryes, R.; Zelnik, L. ASAP: Architecture Search, Anneal and Prune. In Proceedings of the Proceedings of the International Conference on Artificial Intelligence and Statistics, 2020.

15. Liang, H.; Zhang, S.; Sun, J.; He, X.; Huang, W.; Zhuang, K.; Li, Z. DARTS+: Improved differentiable architecture search with early stopping. arXiv preprint arXiv:1909.06035 2019.
16. Chu, X.; Wang, X.; Zhang, B.; Lu, S.; Wei, X.; Yan, J. DARTS-: robustly stepping out of performance collapse without indicators. arXiv preprint arXiv:2009.01027 2020.
17. Krizhevsky, A.; Nair, V.; Hinton, G. CIFAR-10 (Canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html 2010, 5, 4.
18. Krizhevsky, A.; Nair, V.; Hinton, G. CIFAR-100 (Canadian Institute for Advanced Research). URl: https://www. cs. Toronto. edu/kriz/cifar. html 2014.
19. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2009.
20. Chu, X.; Zhou, T.; Zhang, B.; Li, J. Fair darts: Eliminating unfair advantages in differentiable architecture search. In Proceedings of the Proceedings of the European Conference on Computer Vision, 2020.
21. Peters, B.; Niculae, V.; Martins, A.F. Sparse Sequence-to-Sequence Models. In Proceedings of the Proceedings of the Annual Meeting of the Association for Computational Linguistics, 2019