OPT2023: 15th Annual Workshop on Optimization for Machine Learning

FaDE: Fast DARTS Estimator on Hierarchical NAS Spaces

Simon Neumeyer NEUMEYER.SIMON@GMX.DE
Julian J. Stier JULIAN.STIER@UNI-PASSAU.DE
Michael Granitzer
University of Passau, Germany

Abstract

Vast search spaces and expensive architecture evaluations make neural architecture search a challenging problem. Hierarchical search spaces allow for comparatively cheap evaluations on neural network sub modules to serve as surrogate for architecture evaluations. Yet, sometimes the hierarchy is too restrictive or the surrogate fails to generalize. We present FaDE that uses differentiable architecture search to iteratively obtain relative performance predictions on finite regions of a hierarchical neural architecture search space. The relative nature of these ranks calls for a memory-less, batch-wise outer search algorithm which we provide in form of an evolutionary approach that features a pseudo-gradient descent. FaDE is especially suited on deep hierarchical respectively multi-cell search spaces which it can explore by linear instead of exponential cost and therefore eliminates the need for a proxy search space. FaDE solely trains on the neural architecture search space, not on any space of neural architecture sub modules. Our experiments show that firstly, FaDE-ranks on finite regions of the search space correlate with corresponding architecture performances and secondly, the ranks can empower a pseudo-gradient evolutionary search on the complete neural architecture search space.

Keywords: darts, hierarchical space, neural architecture search, automl, differentiable structure optimization

1. Introduction

Automatically finding structures of deep neural architectures is an active research field. The exponentially growing space of directed acyclic graphs (DAGs), their complex geometric structure and the expensive architecture performance evaluation make neural architecture searches a challenging problem. Methods such as evolutionary and genetic algorithms, bayesian searches and differentiable architecture searches compete for the most promising automatic approaches to conduct neural architecture searches [3].

Differentiable architecture search (DARTS) is a successful and popular method to relax the search space into a differentiable hyper-architecture. This relaxation allows to use differentiable search methods to learn both model weights and architectural parameters to evaluate sub-paths of a hyper-architecture [5]. While the combined and weight-shared hyper-architecture allows for a very fast training of few GPU days, the search space is limited to subspaces of the defined hyper-architecture. Evolutionary searches, in contrast, are way more dynamic in the way they restrict the search space. Without any weight-sharing tweaks, this usually comes with a significant higher computational time.

Common NAS spaces are hierarchical in the sense of featuring repeated neural sub-modules consisting of some layers of neurons [9]. Optimization algorithms on such search spaces often only

train a single neural sub-module, potentially on a shallower proxy domain, and use these results to construct a target architecture. However, the search results do not always generalize well to the real search space [1].

We present a *FAst Darts Estimator on hierarchical search spaces (FaDE)* that aims at optimizing chained like hierarchical architectures while not resorting to a proxy domain. This is done by iteratively fixing a finite set of sub-module architectures per neural sub-module and using differentiable architecture search to estimate the rank, i.e. the relative performance within this iteration, of any possible resulting overall architecture. As the estimations are of relative nature, we require a state-less, batch-wise optimization algorithm to determine from those estimations a new finite set of cell architectures per cell. To this end, we apply an evolutionary approach which incorporates a pseudo-gradient descent for candidate generation. FaDE runs one independent optimization algorithm per cell which allows it to optimize additional depth with linear instead of exponential cost.

Our contributions contain the first usage of differentiably obtained ranks for neural architecture search in an open-ended search space. The usage is justified with a correlation analysis. We provide code and data of our experiments for reproducibility in a github repository. The presented approach might be generalized to further types of hierarchical search spaces and could also be employed with other state-less, batch-wise search strategies in open-ended search spaces

2. Fast DARTS Estimator on Chained Hierarchical NAS Spaces

We introduce FaDE on chained hierarchical search spaces to predict the relative performance of architectures within a finite region of the search space and iteratively let the obtained FaDE-ranks guide a search on the complete search space. The structure of finite regions is not arbitrary, but bounded to the set of architectures contained in a hyper-architecture. We use DARTS to train such a hyper-architecture, FaDE to predict the corresponding region of the search space from a trained hyper-architecture, and a mapping of the search space into euclidean space together with a pseudo gradient descent to guide the exploration of the search space.

2.1. Chained Hierarchical NAS Space

Motivated by repeated motifs in hand-crafted architectures, [9] introduce hierarchy to NAS search spaces by considering an architecture to be constructed from several structurally identical neural architecture sub-modules, so called *cells*. Cells feature the same neural network architecture, but each cell has its proper weights. A cell typically consists of several convolutional layers with variable types of convolutions and variable connections between layers. Hence, optimizing the *cell architecture* is often equivalent to finding the type of convolutional operation for a fixed number of layers and determining which layers are being connected. The *macro-architecture* determines how multiple cells are stacked successively. Many search strategies on such search spaces solely optimize the architecture of a single cell which is often done on a shallower *proxy domain*. In image classification, such a proxy domain might consist in reducing the complexity from *CIFAR-100* to *CIFAR-10*. Even though such search strategies can achieve benchmark results in small time, they are not always successful. On the one hand, the performance of a single cell might not generalize to the performance of an overall neural network architecture consisting of multiple structural copies of that cell. Further, a search space consisting solely of structural copies of a single cell architecture might be too restrictive [1].

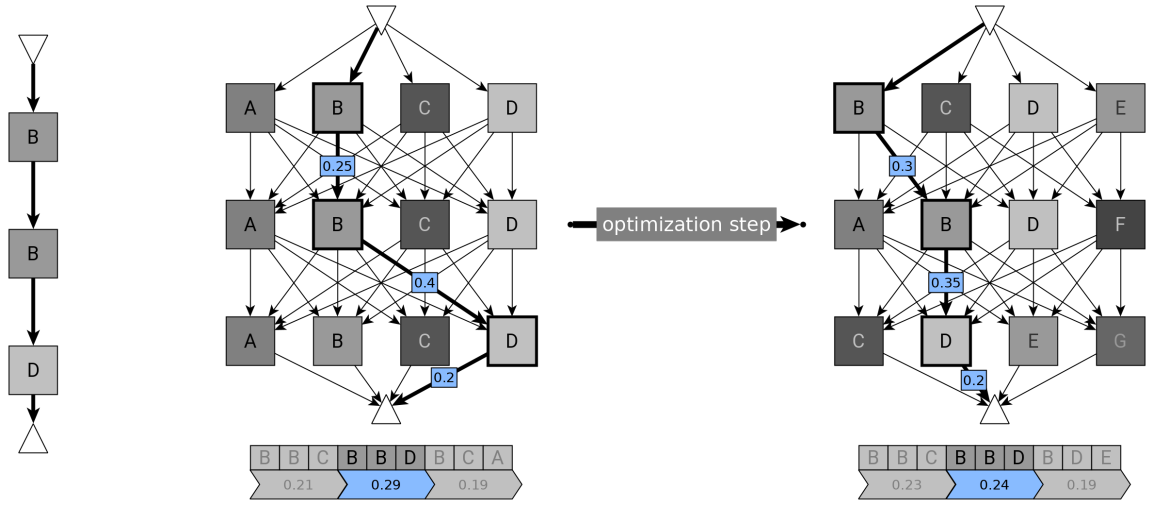


Figure 1: (left) Discrete architecture *BBD* $\in \mathcal{S}^3$ featuring cell architectures $B, D \in \mathcal{S}$. (middle) *BBD* contained in a hyper-architecture $H \in \mathcal{H}_{3,4}(\mathcal{S})$ that allows for several cell architectures per row. Obtaining relative FaDE-ranks on trained hyper-architecture: factorizing architecture parameters along the corresponding path of the hyper-architecture. (right) Each step in the outer NAS optimization discovers new cell architectures per row.

2.2. Training a Chained Hierarchical Architecture using DARTS

For an abstract space of cell architectures $\mathcal{S}$ and a depth $d \in \mathbb{N}$, we build the chained search space $\mathcal{S}^d$, see Figure 1. Given a window size $w \in \mathbb{N}$, we consider the corresponding space of hyper-architectures as $\mathcal{H} := \mathcal{H}_{d,w}(\mathcal{S}) := \mathcal{S}^{d \times w}$, see Figure 1 (middle). We use a matrix notation for hyper-architectures for convenience. Note, that any hyper-architecture $H \in \mathcal{S}^{d \times w}$ can be identified with a subset of the search space by considering $\times_{i \leq d} \{H_{ij} \mid j \leq w\} \subset \mathcal{S}^d$, the cross product modelling the combinatorics of chaining cells along the depth of the search space. Hence, a row of $H \in \mathcal{S}^{d \times w}$ represents the pool of cells that H features at the corresponding depth.

Using differentiable architecture search (DARTS) [5], we endow a hyper-architecture $H \in \mathcal{H}$ with architecture parameters $\alpha \in [0,1]^{d \times w}$, such that after training H with a bi-level optimization algorithm, architecture parameter α_{ij} reflects the performance of cell H_{ij} in the context of exclusive competition within each row of H. For convenience, $\alpha_i, i \leq d$, always denotes the architecture parameters after the Softmax transformation.

In our work, the cells themselves serve as building blocks for DARTS as opposed to [5] where DARTS is being applied to optimize the architecture of a single cell. Following [2] and [8] we choose Gumbel-Softmax [4] to incorporate the architecture parameters into the computation path. Usually, DARTS-based methods consider the architecture parameters as variables over a convex loss surface and optimize them in the same fashion as neural networks weights. We use a constant learning rate and manually regularize the architecture parameters such that we prevent the hyper-architecture from converging too early while pushing such a convergence in later epochs. To

this end, we add the maximum norm on the Gumbel-Softmax regularized architecture parameters, weighted with a linearly decreasing scalar, to the loss function.

Weight Sharing and Cell-dependent Regularization In addition to the implicit weight sharing within the hyper-architecture, implied by a linear number of cells making up for an exponential number of contained architectures, weights might also be shared per row, which yielded good results during our experiments. An interesting trait of our hyper-parameter is its possibility to control the architecture learning speed per stage. To this end, we considered an individual architecture parameter regularization scalar per stage. Our metric assessing the surrogate capabilities of our hyper-architecture was higher when we let architecture learning happen slower for deeper stages.

2.3. Deriving FaDE-ranks on Hyper-Architecture

We use α to rank the subset of architectures contained in H based on

$$\psi_\alpha : H \to \mathbb{R}, (H_{1k_1}, \dots, H_{dk_d}) \mapsto \prod_{i \leq d} \alpha_{ik_i}$$

as shown in Figure 1 (middle). Note that ψ_α is one of many ways to apply the information encoded in α to a ranking of corresponding architectures. The benefit of this approach arises from the practical - not theoretical - assumption of independence along the depth of an architecture, that allows for predicting an exponential search space in linear time. Training several architectures $H_{val} \subset H \subset \mathcal{S}^d$ from scratch yields a validation function ρ on H_{val}. A rank correlation coefficient between ρ and ψ_α, the latter restricted to the validation set, documents how well ψ_α predicts relative performances of single architectures contained in H.

2.4. Joint Batch-wise Pseudo Gradient Descent on a Chained Hierarchical Search Space

A proper correlation between ρ and ψ_α indicates the usefulness of the information contained in the α parameters and motivates us to use the latter in guiding a memory-less, batch-wise search in $\mathcal{S}^d$. The downside of the hyper-architecture approach persists in the relative nature of the evaluation, implying that FaDE-ranks in general can not be compared among different hyper-architecture evaluations. In contrast to the approach in [6], we argue though, that changes in the normal weights of a hyper-architecture eventually invalidate former architecture evaluations within that hyper-architecture and hence any information obtained, whether prediction or real evaluation, inhibits a relative nature anyways, its information content *fading* during search. Search methods such as gradient descent do not need memory and hence can still use the relative information, in our case the FaDE-ranks, to navigate a neural architecture search. To apply gradient descent, we assume the search space to be euclidean and use finite differences on a batch of FaDE-ranks to approximate a gradient. Details on the caveat of $\mathcal{S}$ to be euclidean are being discussed further below.

The overall search works by iteratively sampling hyper-architectures $H^{(t)}, t \in \mathbb{N}$ where the i-th row of $H^{(t+1)}$, $i \leq d$ is solely dependent on the i-th row of the corresponding architecture parameter $\alpha^{(t)}$, obtained after training $H^{(t)}$. Compare Figure 1. Thus, formally, we consider d independent w-dimensional stochastic processes in $\mathcal{S}^w$ and we therefore refer to the search over $\mathcal{S}^d$ being a joint search over $\mathcal{S}$. The goal is to find hyper-architectures containing well-performing single architectures. For any row $i \leq d$, an *anchor* point $M_i^{(t)} \in \mathcal{S}$ is being maintained, with $M_i^{(1)}$ being randomly initialized. The cells of row i in $H^{(t+1)}$ are being derived from the standard unit

vectors originating from the anchor:

$$H_i^{(t)} = \{M_i^{(t)}\} \cup \{M_i^{(t)} \pm \gamma * e_k \mid k \leq \frac{w}{2}\}$$

where the dimension of $\mathcal{S}$ is assumed to be $\frac{w}{2}$ and where e_k denote the standard unit vectors, $k \leq \frac{w}{2}$. The hyper-parameter $\gamma > 0$ controls the width of the local environment around the anchor. After having obtained the architecture parameters for $H_i^{(t)}$ by training $H^{(t)}$, the anchor $M_i^{(t+1)}$ is being derived from descending $M_i^{(t)}$ according to the finite differences along the standard unit vectors:

$$M_i^{(t+1)} = M_i^{(t)} - \lambda \sum_{k=1}^{\frac{w}{2}} e_k(\beta_i(M_i^{(t)} + \gamma * e_k) - \beta_i(M_i^{(t)} - \gamma * e_k)) \tag{1}$$

where $\lambda > 0$ controls the step size of the gradient descent and $\beta_i(\cdot)$ mapping sub-modules to their corresponding architecture parameter after training iteration i.

Note that weight sharing could be considered among successive hyper-architectures. We tested pre-initializing the normal neural network weights of $H^{(k+1)}$ with the trained weights of $H^{(k)}$ which did not yield significantly different results.

Search Space In this work we focus on the graph attributes of neural network cell architectures. We map a directed acyclic graph to a cell architecture by first prepending an input vertex to vertices with no incoming edges and appending an output vertex to all vertices with no outgoing edges. The input vertex just serves as interface to distribute the input vector $\boldsymbol{x}$ to all source vertices. On all edges, except those originating from the input vertex, we place structurally identical convolution layers. Vertices with more than one input edge combine their inputs by summation and ReLU non-linearity. While channel count and feature map size within a cell are being fixed, between succeeding cells we approximately double the channel count while proportionally reducing the feature map size. To enable the pseudo gradient descent from subsection 2.4, we further provide a low-dimensional euclidean *feature space* into which we map the graphs according to a hand-crafted set of numeric graph attributes with same cardinality as the feature space dimensions.

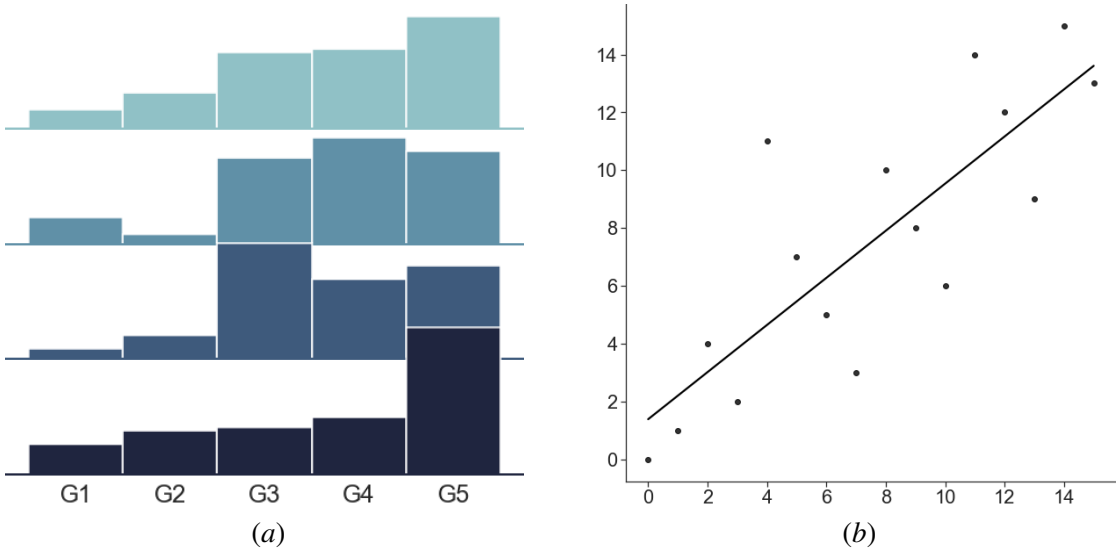
3. Experiments

We consider a multi-cell search space consisting of $n_c = 4$ chained cells, each cell featuring a DAG with less than $n_v = 6$ nodes as cell architecture. In a first experiment we obtain FaDE-ranks on a single hyper-architecture and show that they correlate well with the actual performances of a small subset of architectures contained in the hyper-architecture. Another experiment iteratively trains hyper-architectures according to subsection 2.4 in order to search the complete search space. For the latter we present that search results improve over iterations, though not significantly.

Hyper-Parameters All experiments are conducted on the CIFAR-10 image classification dataset. The employed dataset split is $1 - 1 - 4$, meaning one part was used for *testing*, one part for *architecture training* and four parts for *weight training*. We consider succeeding convolutional cells with ReLU activation, 5×5 kernel size and intermediate max pooling for downsampling. Channel count increases exponentially between cells with 16 or 32 in the final cell. We further feature Cross-Entropy loss, Kaiming weight initialization, a standard normal architecture parameter initialization. The network is optimized in batches of 128 with Adam ($\beta_1 = 0.1, \beta_2 = 10^{-3}, \alpha = 10^{-3}, \epsilon = 10^{-8}$), weight decay ($10^{-4}$)and gradient clipping (10 absolute). We use DARTS with hard Gumbel-Softmax (temperature 10).

3.1. Validating FaDE-Ranks

We construct a hyper-architecture that features the same set of $n_g = 5$ manually selected DAGs as parallel computation paths per cell. Once embedded, each DAG comes with the same number of neural network weights, approximately $0.25 \cdot 10^6$. We obtain a finite search space containing $n_g^{n_c} = 5^4 = 625$ architectures. After training the hyper-architecture, we obtain FaDE-ranks as described in section 2.3. To this end, we consider the architecture parameter distributions per cell, averaged over several experiment repetitions, as independent marginals of a joint discrete distribution on the finite architecture search space. Fig. 2(*a*)subfigure illustrates the predicted marginal distributions of the trained hyper-architecture. Training a subset of 16 manually selected discrete architectures enables calculating a spearman rank correlation between FaDE-ranks and evaluation ranks on the subset, as visualized in fig. 2(*b*)subfigure via linear regression. The spearman rank correlation coefficient of 0.8 is significant and shows that our methodology predicts the relative performances within the small architecture subset quite well. That means, the obtained FaDE-ranks can be used to guide an open-ended search with local information.



(*a*) (*b*)

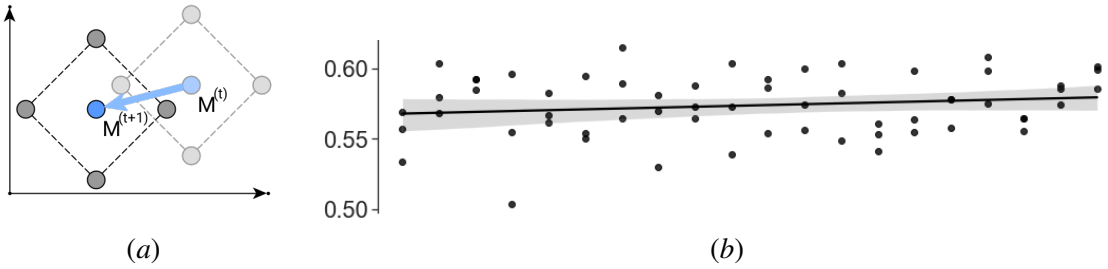
Figure 2: (left) Density of softmaxed architecture parameters: Predict ranks based on averaged architecture parameter per graph architecture (randomly indexed) per cell (dark=deep, light=shallow) (right) Correlation between predicted ranks and evaluated ranks

From further experiments we observed that weight sharing decreases the obtained correlation compared to the results in Figure 2. However, weight sharing yields quite significant correlation results already with a few number of training epochs. This may be owed to the fact, that weight sharing implies weights to be trained more often and thus counteracts the reduction in epochs. We will resort to a fewer number of training epochs while sharing weights in later experiments.

3.2. NAS on iterative FaDE-Ranks

We aim at iteratively improving the cell architectures of the hyper-architecture from subsection 3.1 according to the methodology described in subsection 2.4. A pseudo gradient descent serves as optimization strategy on the three-dimensional euclidean feature space into which we embed all DAGs up to a vertice count of 6. Building on [7], to determine the embedding of a DAG, we choose its eccentricity variance, degree variance and number of vertices, normalized to $[0, 1]$ among all considered DAGs.

A single experiments runs with 100 epochs of gradient descent iteration, each featuring five epochs of hyper-architecture optimization. For each cell $i = 1, \dots, 4$, we obtain feature space trajectories $(M_i^{(t)})_{t=1,\dots,100}$. In order to validate these trajectories, in regular intervals of t, we repeatedly construct an architecture from the DAGs generated from $M_i^{(t)}, i = 1, \dots, 4$, and evaluate it from scratch. We thus obtain a validation function from the trajectory index space into $\mathbb{R}$. An increasing function, coarsely measurable by a linear regression on its index, would indicate our search strategy to yield better architectures over time. 3(*b*)subfigure depicts the discrete model evaluations, including a linear regression. Subsequently, Figure 4 shows per cell from shallow to deep, the trajectories in all three dimensions, while the bottom graphic shows the trajectories aggregated over cells.



(*a*) (*b*)

Figure 3: (left) Depiction of an outer optimization step with gradient descent with finite differences in $\mathcal{S}$ respectively $\mathcal{F}$. Compare Equation 1 for the update of an anchor point $M^{(t)}$ to $M^{(t+1)}$. Any memory-less search strategy can make use of FaDE-ranks by making a step towards better ranked architectures. (right) Evaluation accuracy of architectures generated from points in the search trajectory: evaluation performance is increasing

The obtained pearson coefficient of 0.16 for the linear regression is barely significant. We notice that the trajectories do not alter much after the first 10 epochs, indicating the number of 100 being set too high. However, regression results on the first 10 epochs only are not significant either. Further, the variance in discrete model evaluations seems to be large. Even though, this indicates more profound instabilities in both the underlying feature space and the influence of the architecture on the respective model evaluation, more evaluation repetitions might already mitigate that issue. Note that convergence in *number of operations* towards the high end of the scale occurs in every cell. This convergence is in accordance with what one would naturally expect. It can also be observed that the deeper the cell the more divergent its trajectories. There are artifacts that could be attributed to the sparsity of our search space or poor heuristics of our search space sampling, for example the large step sizes of the trajectories and its occasional chainsaw pattern.

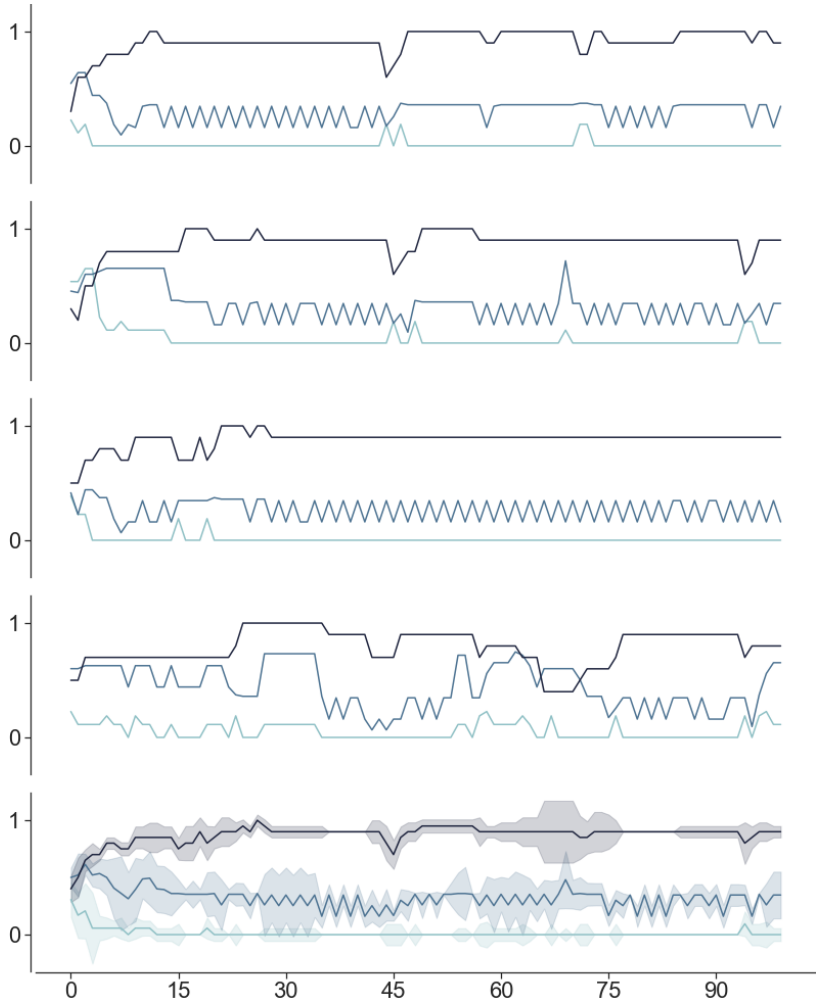


Figure 4: Search trajectories in $\mathbb{R}^3$ per cell per dimension for 100 epochs

4. Conclusion & Future Work

We presented FaDE, a method to leverage differentiable architecture search to aggregate path decisions from a fixed hierarchical hyperarchitecture into point estimates for an open-ended search. The aggregated estimates are called FaDE-ranks and show a positive rank correlation with individually trained architectures. Justified with this correlation, FaDE-ranks can be used to guide an outer search in a pseudo-gradient descent manner. The method is generalizable in a way that alternative strategies for the outer search can be employed as long as the relative nature of the ranks are respected. We see future work in both **1/** the analysis of the quality rank information for global search as well as **2/** experiments with more complex graph feature spaces, e.g. obtained from generative graph models.

References

[1] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1294–1303, 2019.

[2] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[4] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[5] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[6] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[7] Julian Stier and Michael Granitzer. Structural analysis of sparse neural networks. *Procedia Computer Science*, 159:107–116, 2019.

[8] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *International Conference on Learning Representations*, 2018.

[9] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.