Growth strategies for arbitrary DAG neural architectures

Stella Douka, Manon Verbockhaven, Théo Rudkiewicz, Stéphane Rivaud, François P. Landes, Sylvain Chevallier, Guillaume Charpiat

Inria TAU team, LISN, Université Paris-Saclay, Orsay, France

Abstract. Deep learning has shown impressive results, obtained at the cost of training huge neural networks. However, the larger the architecture, the higher the computational, financial, and environmental costs during training and inference. We aim at reducing both training and inference durations. We focus on Neural Architecture Growth, which can increase the size of a small model when needed, directly during training using information from the backpropagation. We expand existing work and freely grow neural networks in the form of any Directed Acyclic Graph. We design strategies that reduce excessive computations and steer network growth toward more parameter-efficient architectures.

1 Introduction

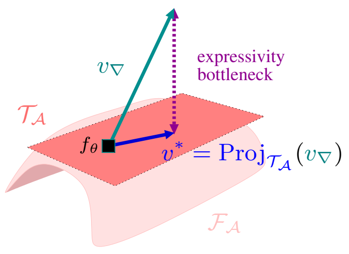
Method	GPU days	kWh
Firefly	1.5	9
NORTH	0.4	2.4
ENAS	0.45	2.7
DARTS	1.5	9

Table 1: GPU power consumption **estimation** on CIFAR-10, assuming 250W power draw.

A common practice to train a deep architecture on a novel problem is to rely on overparametrization – meaning overly wide and deep networks – as it facilitates optimization and yields better results. While it is possible to start with small models that are faster to train, they often lack expressivity and bear optimization issues. Hence, most literature focuses on training large neural networks and then using pruning, distillation, or compression to reduce energy consumption in the inference phase. This includes training the large models and fine-tuning them, requiring a tremendous amount of computational power and training time. On the contrary, Neural Architecture Search methods (NAS) usually train multiple architectures from a finite set and choose the one that performs the best, usually with a trade-off between accuracy and cost. This is extremely resource-consuming and even Differential Architecture Search requires 1.5 GPU days to train on CIFAR-10 [5] (see Table 1). This is where Neural Architecture Growth comes at hand. The idea is to start with the simplest possible neural network and grow it by adding neurons in existing layers or adding entirely new layers, according to the information brought by the backpropagation. Such information can indeed be used to go beyond the usual limitations in small network training, to tackle potential optimization and expressivity issues. The GradMax approach [2], requires initializing all new input weights to zero, thus preserving the function's output. In NORTH [6], one measures the redundancy of the network as the orthogonality between post-activations. In Firefly [8], there is a choice between splitting existing neurons or creating new ones, which also includes adding new layers. All changes are kept local and one decides where to grow by solving the steepest-descent optimization problem. In the approach of [7], some of us introduce the notion of expressivity bottleneck to solve optimization issues in a sequential architecture by increasing its layers' width during training. We refer to [1] for a more thorough review of methods that help improve energy efficiency.

In this paper, we extend the work of [7] by adding new layers on the fly, thus being able, for the first time, to grow neural networks in the form of any Directed Acyclic Graph. We test different strategies to grow the network efficiently and reduce energy costs.

2 Methodology


v_∇
expressivity bottleneck
$\mathcal{T}_\mathcal{A}$
f_θ
$v^* = \text{Proj}_{\mathcal{T}_\mathcal{A}}(v_\nabla)$
$\mathcal{F}_\mathcal{A}$
Fig. 1: Expressivity Bottleneck

Expressivity bottleneck. In Figure 1 we present the concept of *expressivity bottleneck* [7]. We define the manifold $\mathcal{F}_\mathcal{A} := \{f_\theta \mid \theta \in \Theta_\mathcal{A}\}$ as the functional space parameterized by a neural network, that is, the set of all possible functions one can represent by instantiating parameters of a fixed architecture $\mathcal{A}$. The tangent space at f_θ, namely $\mathcal{T}_\mathcal{A} := \left\{\frac{\partial f_\theta}{\partial \theta}\delta\theta \mid \text{s.t. } \delta\theta \in \Theta_\mathcal{A}\right\}$, consists of all the possible functions one can reach on the current manifold $\mathcal{F}_\mathcal{A}$ using small parameter updates, *e.g.* by gradient descent. Now, let us denote by v_∇ the desired update for the function f_θ when we are not constrained by the current architecture: $v_\nabla(x) := -\nabla_{f_\theta(x)}\mathcal{L}(f_\theta(x), y(x)) := -\nabla_a \mathcal{L}(a, y(x))\big|_{a=f_\theta(x)}$. This is the functional gradient, *i.e.* the gradient of the loss w.r.t. the output of the network. The best update we can perform with the current architecture $\mathcal{A}$ is the projection v^* of that desired update onto the tangent space $\mathcal{T}_\mathcal{A}$. As a result, the residual that should be completed by extending the network is: $v_\perp := v_\nabla - v^*$, where $v^* := \text{Proj}_{\mathcal{T}_\mathcal{A}}(v_\nabla) := \arg\min_{v \in \mathcal{T}_\mathcal{A}} \mathbb{E}_{(x,y)\sim\mathcal{P}}\left[\|v_\nabla(x) - v(x)\|_F^2\right]$ and the residual's norm $\Psi := \|v_\perp\|$ is named the *expressivity bottleneck* of the architecture.

One can add more neurons to a hidden state to mitigate the expressivity bottleneck at a given layer, thus growing the network. For further details, we refer the reader to the original paper.

Growing an arbitrary DAG. The contribution of this paper consists in the extension of the work by [7] to non-sequential networks, in the form of any Directed Acyclic Graph (DAG) of fully connected layers. The graph in Figure 2 shows an example of a non-sequential network where every edge represents a fully connected layer and each node represents a hidden state (or addition thereof). We optimize the new weights α, ω so as to decrease the expressivity bottleneck:

$\alpha^*, \omega^* = \arg\min_{\alpha,\omega} \| \omega\, \sigma(\alpha \cdot x) - v_\perp \|$. With the current setting, we can create a network starting from an empty graph, rather than needing to choose a starting point. At each growth step, we have the option to add a direct edge (1 layer), add a new node (with 2 edges, *i.e.* 2 new layers), or increase the size of an existing node by adding new neurons to its input and output layers (increase width). Expanding a node or adding a new one is the same process, as we only need to specify the input and output edges of the new neurons to be added. The peculiarity of this case lies in the fact that the best possible updates $v^*(x)$ of a specific node should take into account at least all the parameters contributing directly to this node. For reference, in Figure 2, when calculating $v^*(x)$ at the hidden state $i+1$, we consider the pre-existing weights W_2 and W_3.

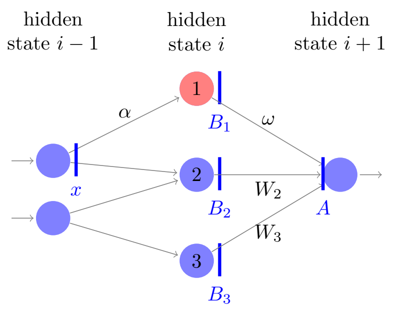


Fig. 2: Example of DAG Network. We assume a new node added in color red with new weights α and ω. We define pre-activities as A and post-activities as B or x.

We split our training dataset into 3 equal parts named ***train-opt***, ***train-ls*** and ***train-gr***. At each growth step, we consider all possible network expansions. For every possibility we optimize the new candidate neurons' weights using *train-opt*. Then, for each new possible direction, we correct its amplitude by minimizing the loss on *train-ls*. Finally, we keep only the best possible expansion, according to an estimate of the loss on *train-gr*, and discard all the others. We then train this newly expanded architecture with SGD, using the concatenation of *train-opt* and *train-ls*, which we refer to as ***inter-train***.

Strategies for Growth. The ***whole search space*** described above is a greedy strategy, where we let the network grow freely based on the *train-gr* loss alone. The search space inflates very fast with the growth steps, together with the associated GPU energy consumption. The ***bottleneck restricted space*** strategy attempts to reduce this space by restricting the available network expansions. In this strategy, we find the node with maximum expressivity bottleneck $A^* = \arg\max_A \Psi_A$ and evaluate only the expansions that contribute to this pre-activity, that is, expanding or adding new layers that output to A^* or expanding the node A^* itself. This way we greatly reduce the search space and thus the search time and GPU energy consumption. In a third strategy, we aim at a trade-off between performance and complexity, within the bottleneck restricted search space. We use the Bayesian Information Criterion as $\text{BIC} = k\log(n) - 2\log(\mathcal{L})$, where k is the number of parameters and n is the sample size. This strategy is named ***BIC + restricted space***. To compare these strategies, we consider an ideal situation where we already know the perfect architecture for the task, that is a Teacher's architecture, thought of as an oracle.

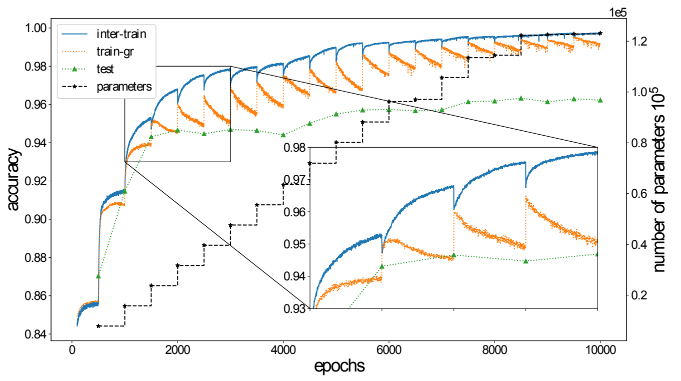


Fig. 3: Neural Architecture Growth results on MNIST using arbitrary DAG networks. We grow the architecture every 500 epochs.

3 Experiments and Results

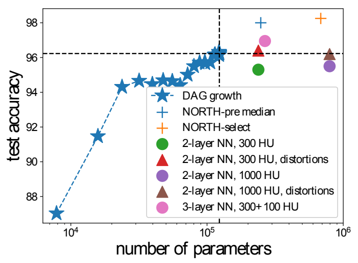


Fig. 4: Baselines on MNIST for test accuracy and number of parameters.

Proof of concept. In [7], we evaluated growing networks on CIFAR-100 with sequential architectures. This study considers further experiments with DAG networks. As we have implemented only fully-connected layers so far, we are conducting a first evaluation on MNIST. To the best of our knowledge, there are no published results of NAS methods on MNIST except for [6]. For this reason, we use the results of [4] with fully-connected layers as our baselines. We use the *whole search space* strategy and at each growth step, we increase the size of the architecture by 10 neurons. We perform intermediate training between growth steps for 500 epochs and evaluate on the test set. We notice in Figure 3 that this intermediate training pushes parameters towards overfitting, but immediately after the architecture grows, the overfit gap between the *inter-train* and *train-gr* sets is reduced and the network finds itself in a more advantageous position, so it can continue learning. In general, by growing we manage to escape potential local minima that force the training accuracy to converge and we gain significantly more accuracy on *train-gr*. The test accuracy sits just below, since we also overfit on *train-gr* after a few steps, as it is used for the expansion selection. Nevertheless, the gain in test performance is slow but significant. The experiment was run for 20 growth steps, requiring a little less than 7 GPU hours (0.29 GPU days). The final architecture achieves an *inter-train* accuracy of 99.7% and a test accuracy of 96.2%. We could keep growing the architecture for more steps but the improvement in accuracy is not significant and the drain on power consumption and additional complexity are not worth the added efficiency. In Figure 4 we see that we do not achieve state-of-the-art test accuracy but our method is extremely competitive in terms of model complexity and is thus cost-efficient. We expect that strategies such as those presented in the next section may further improve this trade-off.

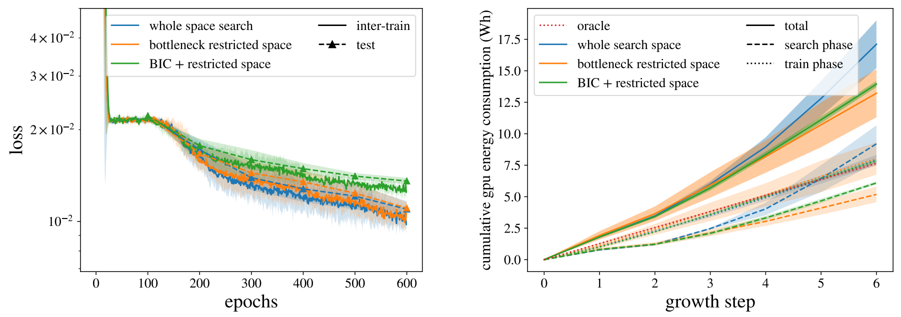


Fig. 5: (a) Teacher-Student loss performance. The highlighted area represents the standard deviation over 6 runs. (b) Teacher-Student cumulative GPU energy consumption in Wh. Oracle: cost to train an architecture identical to the Teacher for the same number of epochs. The highlighted area represents the 85% IQR over 6 runs.

Growth Strategies. To compare strategies in a general framework, we construct a Teacher-Student experiment. We randomly initialize a Teacher network for a regression task, with an input size of 20, two hidden states of size 50, a direct connection from the input to the second hidden state, and selu activations, for a total of 4701 parameters. We can then generate input samples from a uniform distribution and ask this Teacher for labels, to create our train and test datasets. Predicting this output is non-trivial as the intrinsic dimension is the same as the embedding dimension. Indeed, the random initialization of the Teacher parameters and the independent sampling of the input features create many degrees of freedom. We want to compare how our three strategies perform at growing a Student with arbitrary DAG, from scratch, to imitate the Teacher. At each growth step, we can add 10 neurons and we perform intermediate training for 100 epochs. The experiments are shown in Figure 5. We notice a temporary slight drop in performance when we restrict the search space, but it is not a significant one and it disappears after a certain number of epochs. However, GPU energy consumption is decreased by 23% when restricting the search space, for the same loss. We note also that with this strategy, we only consume 70% more energy than the oracle (the ideal scenario where the original Teacher architecture is known and trained from scratch), while here we also have to find the architecture.

Based on these results we estimate that with a grid-search NAS technique instead, we would need to train 7 different architectural structures for 100 epochs to roughly evaluate all the possible DAGs we can achieve for a network of 5 layers. Assuming they would all have the same number of neurons, chosen among a grid of 5, we would have to train and test at least 35 architectures, plus the best architecture fully once, for a total of ≈ 51 Wh. We achieve a more granular result with only 25% of the energy when restricting the search space. In retrospect, reducing the search space based on the bottleneck seems to perform very well in terms of efficiency and cost.

The use of BIC reduces the size of the resulting architecture, with an average of 723 parameters compared to 1479 for the *bottleneck restricted space* and 1454 for the whole search space, but consumes more energy during the search phase than just reducing the space, although with a smaller variance. This is because it tends to choose architectures that create more options for the next search phases. Studying variations on BIC, that may be able to achieve a better performance/architectural complexity trade-off, is left for future work.

4 Conclusion and Future work

In this work, we grow neural architectures in the form of any DAG by adding new layers and direct connections on the fly during training. Our work is based on [7], which introduced the notion of expressivity bottleneck to increase the width of existing layers in a pre-defined architecture structure. Our contribution is to create arbitrary non-sequential fully connected architectures starting from an empty graph, without any predefined structure. We show that our method is competitive in terms of the number of parameters, thus reducing inference time. We compare various strategies to grow an architecture and achieve lower complexity. We manage to reduce the overall training time and thus the GPU energy consumption compared to a grid search among architectures. The next line of research is to further improve our strategy to fulfill an efficient trade-off between performance and complexity. We intend to further extend our work to introduce growable modules for convolutional layers and address scalability.

Acknowledgments This work was supported by grants ANR-22-CE33-0015-01 and ANR-17-CONV-0003 operated by LISN to S.Chevallier and by ANR-20-CE23-0025 operated by Inria to G.Charpiat. The European Union also funded this work under GA no.101135782 (MANOLO project). The development, experiments, and research tools used in this paper consumed roughly 42.5kWh [3], equivalent to 2.18kg of CO2eq in France or flying 105 miles in economy class.

References

[1] Boumendil, A., Bechkit, W., and Benatchba, K. On-device deep learning: survey on techniques improving energy efficiency of dnns. *IEEE Transac. Neural Nets. and Learning Systems* (2024).

[2] Evci, U., van Merrienboer, B., Unterthiner, T., Pedregosa, F., and Vladymyrov, M. Gradmax: Growing neural networks using gradient information. In *ICLR* (2022).

[3] Lannelongue, L., Grealey, J., and Inouye, M. Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science 8*, 12 (2021), 2100707.

[4] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[5] Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *ICLR* (2019).

[6] Maile, K., Rachelson, E., Luga, H., and Wilson, D. G. When, where, and how to add new neurons to ANNs. In *First Conference on Automated Machine Learning (Main Track)* (2022).

[7] Verbockhaven, M., Rudkiewicz, T., Chevallier, S., and Charpiat, G. Growing tiny networks: Spotting expressivity bottlenecks and fixing them optimally. *TMLR* (2024).

[8] Wu, L., Liu, B., Stone, P., and Liu, Q. Firefly neural architecture descent: a general approach for growing neural networks. In *NeurIPS* (2020).